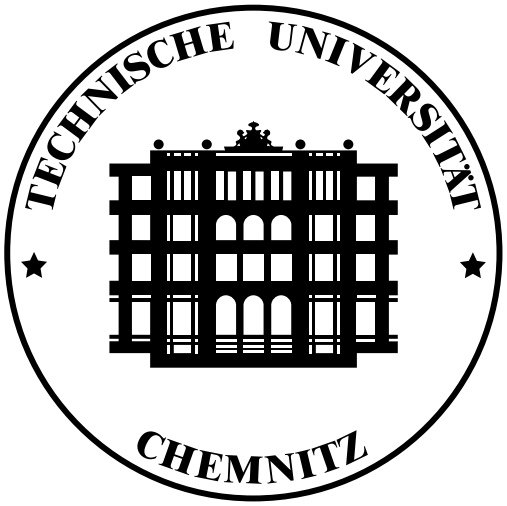


# Analyse und Bewertung von Konfigurationen im Entwurfssystem RODOS

Diplomarbeit
Studiengang Angewandte Informatik

*Autor:*
Marco Fischer
fma@hrz.tu-chemnitz.de
Fakultät für Informatik
TU Chemnitz

*Betreuer:*
Prof. Dieter Monjau
dmo@informatik.tu-chemnitz.de
Fakultät für Informatik
TU Chemnitz

Dipl. Inform. Matthias Sporer
masp@informatik.tu-chemnitz.de
Fakultät für Informatik
TU Chemnitz

Professur Rechnersysteme, Fakultät für Informatik
Technische Universität Chemnitz
27. Februar 2002

**Bibliotheksaufnahme**
Fischer, Marco
"Analyse und Bewertung von Konfigurationen im Entwurfssystem RODOS"
Diplomarbeit, Februar 2002
Fakultät für Informatik, Technische Universität Chemnitz

**Kurzreferenz**

Ziel der Diplomarbeit ist es, Verfahren zur *Analyse* und *Bewertung* von Konfigurationen im Entwurfssystem RODOS (*Reuse-Oriented Design Of Embedded Systems*) zu entwickeln und zu untersuchen. Die *Bewertung* von Konfigurationen wird grundlegend definiert und praktisch in RODOS realisiert. Dazu erfolgt die Einführung nichtfunktionaler *Eigenschaften* und *Anforderungen* für Konfigurationen. Zur *Analyse* von Konfigurationen werden verschiedene *Attributmethoden* entwickelt. Weiterhin wird ein Algorithmus zur Ablaufplanung vorgestellt. Die Resultate der *Analyse* und *Bewertung* werden zum besseren Vergleich verschiedener Konfigurationen in der Arbeitsumgebung von RODOS visualisiert. Die Verfahren finden in einem Fallbeispiel Anwendung.

**Schlüsselwörter**

Wissensbasiertes Konfigurieren, Ablaufplanung, Eingebettete Systeme, Entwurfswerkzeuge

*Technische Universität Chemnitz*
*Fakultät für Informatik*

**AUFGABENSTELLUNG FÜR DIE DIPLOMARBEIT**

***Name, Vorname des Diplomanden:*** Fischer, Marco

***Immatrikulations-Nr.:*** 17594

**Thema:** ***Analyse und Bewertung von Konfigurationen im Entwurfssystem RODOS***

***Zielstellung:***
Statische Analyse und Bewertung der vom Konfigurierer des Entwurfssystems für eingebettete Systeme RODOS aus der Wissensbasis generierten Strukturen (Konfigurationen) unter Einschluss von Komponenten, die geteilte Ressourcen realisieren, bezüglich Zeitverhalten, Kosten, Energieverbrauch, Zuverlässigkeit, Speicherbedarf u. a. (Erfüllung der nichtfunktionalen Anforderungen). Dabei ist zwischen sequentiellen und zyklischen Strukturen zu unterscheiden.

Bereitstellung von Methoden für die Wissensbasis, deren Anwendung die Werte von Attributen hierarchisch höherer Klassen aus den Werten hierarchisch tieferer Klassen berechnet.

Theoretische und experimentelle Untersuchungen zur Ausnutzung von geteilten Ressourcen.

Diverse Auswertungen der Analyseergebnisse für die Komponenten bzw. das gesamte Zielsystem (min/max-Werteintervalle, Auslastungsgrad, Flaschenhälse usw.).

Einführung von Maßen für die Komponenten bzw. die entworfenen Systeme, z. B.
- Grad der Ausnutzung
- 1/(Laufzeit*Kosten) → max
- 1/(Laufzeit*Energieverbrauch) → max

Visualisierung der Analyseergebnisse unter Nutzung verschiedener Diagrammarten (z. B. Gantt-Diagramme für Zeitverhalten und Ausnutzung, Tortendiagramme u. a.)

***Betreuender Hochschullehrer:*** Prof. Dr.-Ing. D. Monjau

***Fakultät:*** Fakultät für Informatik

***Professur:*** Rechnernetze

***Betreuer:*** Prof. Dr.-Ing. Dieter Monjau
Dipl.-Inf. Mathias Sporer

***Beginn am:*** 01.08.2001

***Einzureichen am:*** 31.01.2002

*Verteiler: Prüfungsamt*
*HSL*
*Student (Deckblatt der Diplomarbeit)*
*Betreuer*

*Unterschrift des betreuenden Hochschullehrers*

# Danksagung

Ohne den Rat und die Hilfe Anderer wäre der Abschluss eines Studiums sicherlich um vieles schwerer. So möchte ich an dieser Stelle Herrn Matthias Sporer, Frau Eva Ziegler sowie allen anderen Mitarbeitern der Professur Rechnersysteme für ihre an mich gerichteten Bemühungen danken.
Mein besonderer Dank gilt Herrn Professor Dieter Monjau, der mich nicht nur bei der Anfertigung meiner Diplomarbeit stets tatkräftig unterstützt hat.

Chemnitz, 27. Februar 2002

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

ALAP ..... *As Late As Possible* Wie ASAP, jedoch werden alle Funktionen so spät wie möglich ausgeführt, mit der Bedingung, dass sie eine gegebene Latenzschranke nicht überschreiten.

ASAP ..... *As Soon As Possible* Verfahren zur Bestimmung eines latenzoptimalen Ablaufplanes ohne Ressourcenbeschränkungen. Das Verfahren besteht darin, alle Funktionen so früh wie möglich, aber so, dass alle Datenabhängigkeiten erfüllt sind, auszuführen.

BCET ..... *Best Case Execution Time* Minimale Ausführungszeit einer Funktion.

CAD ..... *Computer Aided Design* Computergestützter Entwurf.

CDFG ..... *Control Data Flow Graph* Kontroll- Datenflussgraph. Ein CDFG ist ein heterogenes Modell, hervorgehend aus der Kombination von Kontrollflussgraphen und Datenflussgraphen.

CLINT++ .... *CLass and Instance NoTation* An der Professur für Rechnersysteme der TU Chemnitz entwickelte deskriptive Sprache zur textuellen Repräsentation der Wissenseinheiten und ihrer Beziehungen.

DAG ..... *Directed Acyclic Graph* Gerichteter, azyklischer Graph. Eine spezielle Klasse von Graphen mit der Eigenschaft, dass

man in ihnen niemals einen Zyklus durchlaufen kann, wenn man die Richtung der Kanten berücksichtigt.

EBNF..... *Extended Backus-Naur Form* Erweiterte Backus-Naur Form. Eine Metasprache zur Definition von Grammatiken.

IDL..... *Information Description Language* Deskriptive Scriptsprache zur Repräsentation von Informationen.

ILP..... *Integer Linear Programming* Ganzzahlige lineare Programmierung. Mathematisches Verfahren zur Lösung linearer Optimierungsprobleme.

RODOS..... *Reuse-Oriented Design Of Embedded Systems* An der Professur für Rechnersysteme der TU Chemnitz entwickeltes Werkzeug zur Unterstützung des wiederverwendungsorientierten Entwurfs eingebetteter Systeme.

WCET..... *Worst Case Execution Time* Maximale Ausführungszeit einer Funktion.

# Symbolverzeichnis

$\beta$ ..... Bindung, eine Funktion $v$ ist an eine Ressource $v_r$ gebunden, wenn $v_r = \beta(v)$

$\beta^{-1}$ ..... Umkehrung der Bindungsfunktion, $V = \beta^{-1}(v_r)$ ist die Menge aller an $v_r$ gebundenen Funktionen

$e$ ..... Kante eines Graphen

$\mathfrak{E}$ ..... Menge aller Eigenschaften einer Konfiguration

$\mathfrak{e}$ ..... Eigenschaft einer Konfiguration

$E$ ..... Menge der Kanten eines Graphen

$E_{back}(v)$ ..... Menge der Rückwärtskanten zum Knoten $v$

$E_{in}(v)$ ..... Menge der eingehenden Kanten eines Knoten $v$

$E_{out}(v)$ ..... Menge der ausgehenden Kanten eines Knoten $v$

$\lambda(v)$ ..... Maximale Anzahl von Ausführungen eines Basisblocks

$L_\tau$ ..... Latenz des Ablaufplanes $\tau$

$\mathfrak{r}(x)$ ..... Anforderung an $x$. Repräsentiert $x$ den Wert einer Eigenschaft $\mathfrak{e}$, so heißt $\mathfrak{r}(x)$ Anforderung an die Eigenschaft $\mathfrak{e}$

$\mathbb{R}$ ..... Menge der reellen Zahlen

$\mathfrak{R}$ ..... Menge aller Anforderungen einer Konfiguration

$\tau$ ..... Ablaufplan, die Startzeit der Funktion $v$ ist durch $t = \tau(v)$ gegeben

$v_S(e)$ ..... Quellknoten der Kante $e$

$v_T(e)$ ..... Zielknoten der Kante $e$

$c(v)$ ..... WCET eines Basisblocks

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einleitung

## 1.1 Motivation

Für einen wirtschaftlichen Erfolg von Produkten der informationsverarbeitenden Industrie ist ein zuverlässiger, schneller und kosteneffektiver Entwicklungsprozess von großer Bedeutung. Seit Mitte der sechziger Jahre des letzten Jahrhunderts hat man erkannt, dass die Komplexität von Informationsverarbeitungssystemen durch exponentielles Wachstum gekennzeichnet ist [Buchenrieder 2001; Ecker 2001]. Der Entwicklungsaufwand für solche Systeme wird ständig größer und der Entwurf immer schwerer beherrschbar. An vielen Stellen wird versucht dem Entwicklungsingenieur durch CAD (*Computer Aided Design*)-Tools die Arbeit zu erleichtern bzw. die effiziente Arbeit überhaupt erst zu ermöglichen. Ausgehend von einer Spezifikation eines zu entwerfenden eingebetteten Systems durch funktionale und nichtfunktionale Anforderungen bestehen eine enorme Anzahl von Entscheidungsmöglichkeiten während des Entwurfsprozesses, deren Auswirkungen nur schwer oder gar nicht vorhergesagt werden können. Diese Entscheidungsvielfalt besteht besonders stark im Bereich des Hardware-Software Co-Designs. Eine Möglichkeit diese Probleme besser zu beherrschen besteht in der Automatisierung von Entwurfsschritten und das Durchsuchen des Entwurfsraumes dem Rechner zu überlassen. Oftmals sind jedoch die Entwurfsräume so komplex, dass spezielles Wissen über die Anwendungsgebiete verwendet werden muss, um das Durchsuchen in sinnvolle Richtungen zu lenken. Um sich "orientieren"

zu können, braucht ein Durchmusterungsmechanismus die Fähigkeit (Teil-) Lösungen zu analysieren und zu bewerten.

Das Entwurfssystem RODOS nutzt anwendungsspezifisches Wissen um Konfigurationen zu erzeugen. Konfigurationen in RODOS stellen (Teil-) Implementierungen von eingebetteten Systemen dar, die die funktionalen Anforderungen erfüllen. Die nichtfunktionalen Anforderungen werden vom Entwurfsingenieur spezifiziert. Ein Konfigurationsalgorithmus sucht dann nach Konfigurationen, die der gegebenen Spezifikation entsprechen, d.h. die auch die nichtfunktionalen Anforderungen erfüllen. Außerdem vervollständigt der Algorithmus unvollständige Konfigurationen schrittweise. Lösungen für das spezifizierte System sind alle vollständigen Konfigurationen, die der Spezifikation genügen, auch gültige Konfigurationen genannt [Förster 2001]. Da die Übereinstimmung der Funktionalität mit der Spezifikation aufgrund des Aufbaus der Wissensbasis immer gegeben ist, sind es die nichtfunktionalen Anforderungen an das System, welche überprüft werden müssen. Dazu ist es notwendig, alle zu beachtenden nichtfunktionalen Anforderungen zu identifizieren und Methoden bereitzustellen, die die entsprechenden Eigenschaften einer Konfiguration bezüglich ihrer Anforderung bewerten.

Diese Methoden müssen einige Merkmale aufweisen, um für die automatische Konfiguration geeignet zu sein. Für eine detaillierte Analyse einer Konfiguration müssen deren Eigenschaften möglichst präzise und verlässlich ermittelt werden. Je genauer Eigenschaften bestimmt werden, desto komplexer und zeitaufwändiger sind die verwendeten Methoden. Beim Konfigurieren ist eine exakte Bestimmung der Eigenschaften nicht von primärer Bedeutung. Natürlich sind für die Lösungen exakte Ergebnisse wünschenswert, aber im Vordergrund steht der nötige Aufwand. So lassen sich mit einer Simulation beispielsweise recht genaue Aussagen über das zeitliche Verhalten eines Systems erzielen, aber beim Durchsuchen des Entwurfsraumes nach einer Lösung würde eine Simulation nach jedem Schritt zu viel Zeit beanspruchen. Daher ist es notwendig, mit gröberen Abschätzungen zu arbeiten, was den Vorteil hat, weniger zeitaufwändige Methoden einsetzen zu können. Solange unterschiedliche Konfigurationen relativ zueinander richtig bewertet werden, ist der Konfigurationsalgorithmus in der Lage, richtig zu entscheiden, und

damit eine Lösung schneller zu finden.

## 1.2 Gliederung

Die vorliegende Arbeit ist folgendermaßen gegliedert. Das erste Kapitel widmet sich der Einführung in die Problematik und der Darstellung von Motivation und Grundlagen. Im zweiten Kapitel wird der theoretische Hintergrund der Bewertung und Analyse von Konfigurationen erläutert und der Bezug zur Literatur hergestellt. Kapitel Drei beschreibt die Umsetzung der Methoden im Entwurfssystem RODOS. Kapitel Vier beinhaltet eine Auswertung der Ergebnisse sowie eine Zusammenfassung und Ausblicke. Wichtige Begriffe sind zur besseren Verständlichkeit im Glossar aufgeführt und beschrieben. Die Bedeutung der verwendeten mathematischen Symbole ist im Symbolverzeichnis beschrieben, benutzte Abkürzungen sind im Abkürzungsverzeichnis aufgeführt. Alle referenzierte Literatur befindet sich im Literaturverzeichnis. Außerdem existieren ein Abbildungsverzeichnis, Tabellenverzeichnis und ein Index zur besseren Auffindbarkeit einzelner Bestandteile der Diplomarbeit. Relevante Teile des Quellcodes und die Wissensbasis des Fallbeispiels befinden sich auf der beiliegenden CD-ROM.

## 1.3 Grundlagen des Entwurfssystems RODOS

Im Folgenden soll ein kurzer Einblick in die relevanten Teile von RODOS und die im Zusammenhang verwendeten Begriffe geliefert werden. Unter [Monjau und Sporer 2000] befindet sich eine detaillierte Beschreibung des Entwurfssystems RODOS.

Zentraler Bestandteil von RODOS sind die *Wissensklassen*. Sie stellen die atomaren Einheiten dar, mit denen das Wissen über eine Domäne[1] formal beschrieben wird. Wissensklassen sind durch verschiedene *Relationen* (s.Tabelle 1.1) miteinander verbunden.

Der Entwurfsablauf unter RODOS wird durch vier *Schichten* unterstützt: die Spezifikationsschicht, Funktionsschicht, Architekturschicht und die Im-

[1] Anwendungsgebiet

| Relation | intra-layer | inter-layer | Beschreibung |
|---|---|---|---|
| is-a | x | | Modelliert die *Verfeinerung* einer Wissensklasse. Ermöglicht die Repräsentation von *Alternativen*. |
| has-parts | x | | Modelliert die *Zerlegung* einer Wissensklasse. Realisiert somit *Aufteilung* in Teilwissen. |
| multirel | | x | Dient zur Abbildung der Spezifikationsschicht auf die Funktionsschicht. |
| architector | | x | Beschreibt die Abbildung der Funktionalität auf eine Architektur. |
| implementor | | x | Beschreibt die Abbildung von Architekturkomponenten auf konkrete Implementierungen. |
| constraints | x | | Bringen *nicht-hierarchische Abhängigkeiten/Bedingungen* zwischen Wissenklassen zum Ausdruck. |

Tabelle 1.1: Die Relationen in RODOS.

plementationsschicht (s. Tabelle 1.2). Sie stellen in dieser Reihenfolge die Bindeglieder zwischen einer informalen Systembeschreibung und der vollständigen Implementierung des Systems dar. Die Schichten beinhalten Netzwerke von Wissensklassen und ordnen ihnen eine Bedeutung zu. So repräsentiert eine Wissensklasse der Spezifikationsschicht einen Bestandteil der Spezifikation eines Systems. In der Funktionsschicht repräsentiert jede Wissensklasse eine Funktion oder Teilfunktion des Systems. Die Architekturschicht beinhaltet Wissensklassen, welche Komponenten einer Architektur repräsentieren. Die konkreten Implementierungen der Architekturkomponenten sind durch Wissensklassen der Implementationsschicht beschrieben. Zusammen stellen die vier Schichten die *Wissensbasis* dar, welche die Grundlage für den Systementwurf mit RODOS bildet.

Jede Wissensklasse hat eine Reihe von Bestandteilen, die je nach Verwendung der Wissensklasse eine bestimmte Bedeutung haben. Einige für die Untersuchungen wichtige Bestandteile sollen kurz beschrieben werden.

Jede Wissensklasse verfügt über eine Menge von *Attributen*. Jedes Attribut hat einen Namen, einen Typ, einen Wert und optional ein Werteintervall.

| Schicht | Beschreibung |
|---|---|
| Spezifikationsschicht | Spezifikationswissen, funktionale und nichtfunktionale Anforderungen |
| Funktionsschicht | Definition und hierarchische Verfeinerung der Systemfunktionalität, alternative Algorithmen/Verfahren |
| Architekturschicht | Architekturbeschreibung, abstrakte Komponenten und deren Verbindungen |
| Implementationsschicht | Beschreibung der Implementierungen der Komponenten der Architekturschicht |

Tabelle 1.2: Inhalt der Schichten in RODOS.

Ein initialer Wert und der Bereich des Werteintervalls sind in der Wissensbasis beschrieben. Jedem Attribut können *Attributmethoden* zugeordnet werden. Eine Attributmethode dient dazu, den Wert eines Attributes neu zu berechnen[2]. Eine Wissensklasse kann aus mehreren Teilen, den *Parts* bestehen. Jeder Part hat einen Namen innerhalb der Wissensklasse und einen Typ, der auf eine andere Wissensklasse verweist. Daher sind die Parts wiederum Wissensklassen, die nur einen Teil der übergeordneten Klasse beschreiben. In der Funktionsschicht repräsentieren Wissensklassen *Funktionen*. Entsprechend stehen Parts einer Wissensklasse für deren Teilfunktionen. Besteht eine Wissensklasse der Funktionsschicht aus mehreren Parts, so muss sie auch einen *Taskgraph* enthalten. Er ist ein Datenflussgraph und beschreibt wie die Teilfunktionen voneinander abhängen. Wissensklassen der Funktionsschicht ohne Parts, welche außerdem nicht weiter spezialisiert werden können, heißen *Basisfunktionen*.

Ein eingebettetes System wird in RODOS durch eine *Konfiguration* beschrieben. Eine Konfiguration besteht aus Klasseninstanzen[3] der Klassen der vier Schichten der Wissensbasis. Eine Konfiguration kann man sich als Ausschnitt aus der Wissensbasis vorstellen. Eine graphentheoretische Definition, in der eine Konfiguration als DAG (*Directed Acyclic Graph*) aufgefasst wird, ist in [Förster 2001] gegeben. In RODOS entstehen Konfigurationen durch Ausführung von *Konfigurationsschritten*. Ein Konfigurationsschritt ist eine

[2] Anstelle der Übernahme des initialen Wertes aus der Wissensbasis.
[3] Abbilder

| Schritt | ver-feinernd | genera-lisierend | Beschreibung |
|---|---|---|---|
| Parametrisierung | x | | Bestimmt und setzt alle Attributwerte einer Wissensklasse. |
| Multirel ausführen | x | | Erzeugt die notwendigen Wissensklassen der Funktionsschicht aus der Spezifikation. |
| Zerlegung | x | | Erzeugt und verbindet die Parts einer Wissensklasse. |
| Spezialisierung | x | | Ersetzt eine Wissensklasse durch einen ihrer is-a Nachfolger. |
| Generalisierung | | x | Kehrt eine Spezialisierung um. |
| Aggregation | | x | Parts werden durch die Wissensklasse ersetzt, der sie angehören. |
| Parametrisierung | x | | Bestimmt und setzt die Werte aller Attribute einer Wissensklasse. |
| Funktionsbindung | x | | Bindet eine Basisfunktion an eine bestimmte Architekturkomponente. |
| Architekturmapping | x | | Wählt eine Architektur aus. |
| Implementierung | x | | Ordnet einer Architekturkomponente eine Wissensklasse aus der Implementationsschicht zu. |

Tabelle 1.3: Inhalt der Schichten in RODOS.

elementare Aktion, die manuell vom Entwurfsingenieur oder bei der automatischen Konfiguration vom Konfigurationsalgorithmus ausgeführt wird. Eine Übersicht über die Konfigurationsschritte gibt Tabelle 1.3. *Verfeinernde* Konfigurationsschritte verändern eine Konfiguration in Richtung Vollständigkeit. Das Gegenteil bewirken *generalisierende* Konfigurationsschritte. Zusätzlich gibt es noch Aktionen wie das Löschen oder Erzeugen von Klasseninstanzen.

Eine Konfiguration heißt *vollständig*, wenn kein verfeinernder Konfigurationsschritt mehr möglich ist. Eine Konfiguration heißt *gültig*, wenn sie die funktionalen und nichtfunktionalen Anforderungen sowie alle Constraints erfüllt. Eine vollständige gültige Konfiguration heißt *Zielkonfiguration* und ist eine Lösung für das spezifizierte System.

# Kapitel 2

# Theoretische Grundlagen der Bewertung und Analyse

## 2.1 Bewertung

Jedes technische System ist durch eine Reihe von Eigenschaften gekennzeichnet. Um eine Eigenschaft quantitativ[1] zu erfassen, muss ein Maß existieren, mit dem unter Verwendung einer Metrik der Eigenschaft ein Wert zugeordnet werden kann. Diese Zuordnung wird als Messung bezeichnet und deren Ergebnis als Produkt von Maßzahl und Maßeinheit angegeben.

**Definition 2.1.1 (Wert einer Eigenschaft)** *Unter dem Wert einer Eigenschaft wird die gemessene Größe in Bezug auf ein bestimmtes Maß verstanden. Es existiere eine Funktion*

$$M(\mathfrak{e}) : \mathfrak{E} \mapsto \mathbb{R} \quad \forall \mathfrak{e} \in \mathfrak{E}, \tag{2.1}$$

*welche der Eigenschaft $\mathfrak{e}$ einen Wert zuordnet. Dabei stelle $\mathfrak{E}$ die Menge aller Eigenschaften dar. Die Funktion $M$ abstrahiert gewissermaßen eine Messung.*

Für alle Eigenschaften, die für den Einsatz des Systems erfüllt sein müssen, existieren in der Spezifikation Anforderungen. Unter einer Anforderung kann man sich daher eine Bedingung über eine bestimmte Eigenschaft vorstellen, z.B. in Form einer mathematischen Ungleichung.

[1] also in Größe, Menge, Stärke etc.

**Definition 2.1.2 (Anforderung)** *Sei $\mathfrak{r}(x)$ eine Aussageform mit einer freien symbolischen Variablen $x$. Repräsentiert $x$ den Wert einer Eigenschaft $M(\mathfrak{e})$, dann heißt $\mathfrak{r}(x)$ Anforderung an $\mathfrak{e}$. Die Menge $\mathfrak{R}$ bezeichne die Menge aller definierten Anforderungen*[2].

**Beispiel 2.1.1 (Anforderung)** *Gegeben sei die Aussageform*

$$\mathfrak{r}(x) : x \text{ ist kleiner als } 500\text{ms}. \tag{2.2}$$

*Steht x für den Wert der Eigenschaft "Antwortzeit", dann bezeichnet man die Aussageform als Anforderung an die Eigenschaft "Antwortzeit".*

Zu bemerken ist, dass eine Anforderung keinen "Wert" besitzt. Um festzustellen, ob ein System die gestellten Anforderungen einhält, müssen die Anforderungen "bewertet" werden. Eine Bewertung benutzt den Wert einer Eigenschaft, um der Anforderung selbst einen Wert zu geben, d.h. die Aussageform in eine Aussage zu transformieren.

**Definition 2.1.3 (Bewertung von Anforderungen)** *Unter Bewertung einer Anforderung $\mathfrak{r}(M(\mathfrak{e}))$ wird die Zuordnung eines Zahlenwertes in Abhängigkeit des Wertes der Eigenschaft $\mathfrak{e}$ verstanden:*

$$B_{\mathfrak{r}}(\mathfrak{r}) : \mathfrak{R} \mapsto W_B \qquad W_B \subseteq [0,1], \tag{2.3}$$

*wobei $W_B$ den Wertebereich der Bewertungsfunktion $B_{\mathfrak{r}}$ bezeichne.*

Nachdem die Bewertung einzelner Anforderungen eingeführt wurde, kann eine Bewertung von Konfigurationen hergeleitet werden. Dazu müssen die Bewertungen der Anforderungen in geeigneter Art und Weise kombiniert werden. Ziel ist es, durch die Verknüpfung der Einzelbewertungen eine Gesamtbewertung für die Konfiguration zu gewinnen, die folgende Merkmale hat:

- normierter Wertebereich, i.e. $[0,1]$
- veränderbare Gewichtung der Anforderungen

[2] Die Bezeichnungen "$\mathfrak{r}$" bzw. "$\mathfrak{R}$" sind von Anforderung (engl. Requirement) entlehnt.

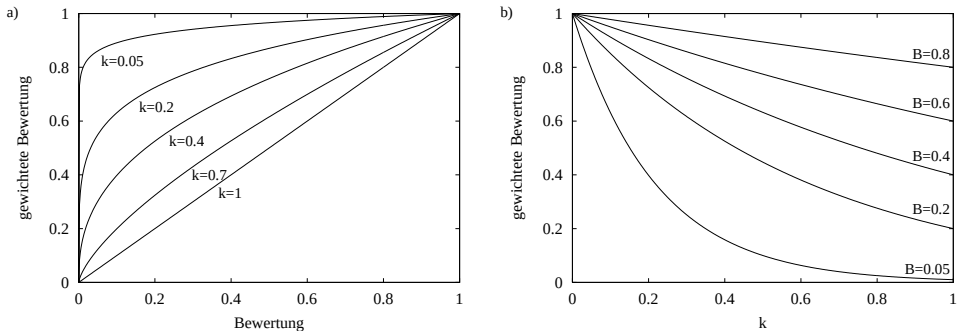


Abbildung 2.1: Wirkung des Gewichtungsfaktors: $B_{\mathfrak{r}}(\mathfrak{r})^k$ für a) verschiedene Gewichtungsfaktoren b) verschiedene Bewertungen in Abhängigkeit von $k$.

**Definition 2.1.4 (Konfigurationsbewertung)** *Gegeben seien die Menge der Eigenschaften $\mathfrak{E}$, die "Messung" der Eigenschaften $M(\mathfrak{e})$, die Menge der Anforderungen $\mathfrak{R}$, und die Bewertungsfunktionen $B_{\mathfrak{r}}$. Die Konfigurationsbewertung ist wie folgt definiert:*

$$B_0(\mathfrak{R}) = \prod_{\mathfrak{r}\in\mathfrak{R}} B_{\mathfrak{r}}(\mathfrak{r})^{k_{\mathfrak{r}}} \qquad k_{\mathfrak{r}} \in [0,1], \tag{2.4}$$

*wobei $\mathfrak{r}$ als $\mathfrak{r}(W(\mathfrak{e}))$ zu verstehen ist. Für jede Anforderung existiert ein Gewichtungsfaktor $k_{\mathfrak{r}}$.*

Über den Gewichtungsfaktor $k_{\mathfrak{r}}$ kann der Einfluss der einzelnen Anforderungen auf die Konfigurationsbewertung verändert werden. Abbildung 2.1 verdeutlicht die Wirkung des Gewichtungsfaktors. Für $k_{\mathfrak{r}} \to 0$ geht der Einfluss gegen Null. Die Gewichtung bewirkt, dass die Bewertung in Richtung 1 verschoben wird und daher weniger auf die Konfigurationsbewertung einwirkt. Aus diesem Grunde wird für den Sonderfall $k_{\mathfrak{r}} = 0$ definiert

$$0^0 \equiv 1, \tag{2.5}$$

so dass auch eine mit 0 bewertete Anforderung durch einen Gewichtungsfaktor von Null keinen Einfluss auf die Konfigurationsbewertung hat. Für $k_{\mathfrak{r}} \to 1$ gehen die gewichteten Bewertungen gegen ihren ursprünglichen Wert.

Mit den oben genannten Einschränkungen sind unterschiedliche Varianten von Bewertungsfunktionen denkbar. Eine natürliche Form der Bewertungsfunktion ergibt sich aus der gebräuchlichen Auffassung von Anforderung:

"Eine Anforderung[3] ist entweder erfüllt, oder sie ist nicht erfüllt."

Das ist das Axiom der zweiwertigen Logik, wodurch sich der Wertebereich der Bewertungsfunktion definieren läßt, indem man zum Beispiel "erfüllt" $\equiv 1$ und "nichterfüllt" $\equiv 0$ erklärt..

**Definition 2.1.5 (Binäre Bewertung)** *Eine Bewertungsfunktion*

$$B_{\mathfrak{r}}(\mathfrak{r}) : \mathfrak{R} \mapsto W_B \qquad mit \quad W_B = \{0, 1\} \tag{2.6}$$

*heißt Binäre Bewertung der Anforderung* $\mathfrak{r}$.

**Beispiel 2.1.2 (Binäre Bewertung)** *Gegeben sei die Anforderung* $\mathfrak{r}(x)$ *aus Beispiel 2.1.1. Die natürliche Auffassung der Anforderung ergibt folgende Bewertung:*

$$\mathfrak{r}(x) \qquad ist\ erfüllt,\ wenn \qquad x < 500ms \tag{2.7}$$

$$\mathfrak{r}(x) \quad ist\ nicht\ erfüllt,\ wenn \quad x \geq 500ms \tag{2.8}$$

*Ersetzt man "erfüllt" mit '*1*, "nicht erfüllt" mit* 0 *und definiert eine Menge* $A = \{x | x < 500ms\}$*, so kann man die binäre Bewertungsfunktion als*

$$B_{\mathfrak{r}}(\mathfrak{r}) = \begin{cases} 1, & x \in A \\ 0, & x \in \bar{A} \end{cases} \tag{2.9}$$

*schreiben*[4]*.*

Bei ausschließlicher Verwendung der binären Bewertung für alle Anforderungen, ergeben sich einige spezielle Eigenschaften für die binäre Bewertungsfunktion der Konfiguration.

**Satz 2.1.1** *Sind alle Bewertungsfunktionen binär, dann gilt*

$$W(B_0(\mathfrak{R})) = \{0, 1\}. \tag{2.10}$$

[3] Im eigentlichen Sinne ist hier die Bewertung der Anforderung gemeint.

[4] Diese Schreibweise erscheint hier etwas umständlich, ist aber von Vorteil bei der Einführung einer anderen Bewertungsfunktion weiter unten.

**Beweis 2.1.1** *Es wird vorausgesetzt, dass alle Bewertungsfunktionen binär sind:*

$$\forall \mathfrak{r} \in \mathfrak{R} : W(B_{\mathfrak{r}}(\mathfrak{r})) = \{0, 1\}. \tag{2.11}$$

*Die Terme der Gleichung 2.4 haben die Form* $x^k$ *mit* $x \in \{0, 1\}$ *für die gilt:*

*I)* $0^k = 0 \quad \forall k_{\mathfrak{r}} > 0$

*II)* $0^0 = 1 \quad$ *(s. Gleichung 2.5)*

*III)* $1^k = 1 \quad \forall k_{\mathfrak{r}} \in \mathbb{R}$

*Daraus folgt* $W(B_{\mathfrak{r}}(\mathfrak{r})^{k_{\mathfrak{r}}}) = \{0, 1\}$. *Die Konfigurationsbewertung besteht folglich nur aus Multiplikationen von* 0 *und* 1, *woraus sich der Wertebereich*

$$W(B_0(\mathfrak{R}) = \{0, 1\} \tag{2.12}$$

*ergibt.* ■

**Satz 2.1.2** *Die Bewertung einer Konfiguration unter ausschließlicher Verwendung binärer Bewertungsfunktionen ist* 1, *genau dann, wenn alle Anforderungen mit* 1 *bewertet werden, oder*

$$B_0(\mathfrak{R}) = 1 \quad \leftrightarrow \quad \forall \mathfrak{r} \in \mathfrak{R} : B_{\mathfrak{r}}(\mathfrak{r}) = 1. \tag{2.13}$$

**Beweis 2.1.2** *Sobald ein Term des Produktes* 0 *ist, ist das Produkt gleich* 0. ⇒ *Nur wenn alle Terme gleich* 1 *sind ist die Bewertung der Konfiguration gleich* 1.

■

Das Ziel der Bewertung von Konfigurationen in RODOS ist es, ein Maß für die Güte von Konfigurationen zur Verfügung zu stellen. Dieses Maß soll dem automatischen Konfigurationsalgorithmus als Wegweiser bei der Suche dienen. Im Falle einer binären Bewertung kann der Konfigurierer nur zwei Mengen von Konfigurationen unterscheiden. Jene, bei denen alle Anforderungen mit 1 (also erfüllt) bewertet wurden, und jene, bei denen nicht alle Anforderungen mit 1 bewertet wurden. Da es sehr wahrscheinlich ist,

dass dem Konfigurationsalgorithmus zu einem bestimmten Zeitpunkt mehrere Konfigurationen zur Auswahl stehen, welche mit $B_0 = 1$ bewertet wurden, existiert nicht in jedem Fall eine ausgezeichnete Konfiguration, die als nächste ausgewählt werden kann. Der Grund dafür liegt in dem zweielementigen Wertebereich der Bewertungsfunktion (s. Satz 2.1.1). Die Einführung einer Bewertung mit einem Wertebereich höherer Kardinalität könnte Abhilfe für dieses Problem schaffen.

In der Mengentheorie wird eine scharfe Menge über die sogenannte charakteristische Funktion beschrieben[5]

$$\mu_{CA} : A \mapsto \{0, 1\}, \tag{2.14}$$

mit der Bedeutung ein Element gehört zur Menge $A$, wenn $\mu_{CA} = 1$, und ein Element gehört nicht zur Menge $A$, wenn $\mu_{CA} = 0$. Danach kann die binäre Bewertungsfunktion in Beispiel 2.1.2 mit der charakteristischen Funktion definiert werden:

$$B_{\mathfrak{r}}(\mathfrak{r}) = \mu_{CA} \quad mit \quad \mu_{CA} = \begin{cases} 1, & \text{x<500ms} \\ 0, & \text{sonst.} \end{cases} \tag{2.15}$$

Um einer differenziertere Bewertung zu ermöglichen, kann sich nun der Theorie der unscharfen Mengen beholfen werden. Eine unscharfe Menge ist definiert durch die charakteristische Funktion[6] [Jamshidi 1996, S.123-124]

$$\mu_{FA} : A \mapsto [0, 1]. \tag{2.16}$$

Dadurch gehören Elemente der Menge $A$ mit einem Wert zwischen 0 und 1 an. Dieser Wert heißt *Grad der Mitgliedschaft*. Je größer der Wert, desto "mehr" gehört ein Element der Menge an. Die charakteristische Funktion einer unscharfen Menge wird auch *Mitgliedsfunktion* genannt. Abbildung 2.2 zeigt ein Beispiel einer Mitgliedsfunktion im Vergleich zur charakteristischen Funktion einer scharfen Menge. Definiert man die Bewertungsfunktion mit $\mu_F$ anstelle von $\mu_C$, ergibt sich eine reellwertige Bewertungsfunktion.

[5] Der Index setzt sich zusammen aus "C" für scharf (engl. crisp) und der Bezeichnung der mit der Funktion definierten Menge.

[6] Unscharfe Menge, engl. Fuzzy Set. Daher der Index "F".

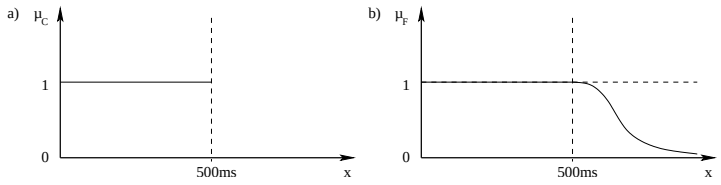


Abbildung 2.2: a) Charakteristische Funktion einer scharfen Menge, b) Beispiel einer Mitgliedsfunktion

**Definition 2.1.6 (Reelle Bewertung)** *Eine Bewertungsfunktion*

$$B_{\mathfrak{r}}(\mathfrak{r}) : \Re \mapsto W_B \qquad mit \quad W_B = [0,1] \tag{2.17}$$

*heißt Reelle Bewertung der Anforderung* $\mathfrak{r}$.

**Beispiel 2.1.3 (Reelle Bewertung)** *Ausgehend von Beispiel 2.1.2 soll eine reelle Bewertungsfunktion abgeleitet werden. Dazu bietet sich die sogenannte L-Funktion an, welche folgendermaßen definiert ist:*

$$L(x,\gamma,\delta) = \begin{cases} 1 & x < \gamma \\ 1-(x-\gamma)/(\delta-\gamma) & \gamma \leq x \leq \delta \\ 0 & x > \delta \end{cases} \tag{2.18}$$

*Eine reelle Bewertung ergibt sich, wenn die Bewertungsfunktion definiert ist als:*

$$B_{\mathfrak{r}}(\mathfrak{r}) = L(x; 500\text{ms}, 1000\text{ms}). \tag{2.19}$$

*Abbildung 2.3 zeigt den Verlauf der Bewertungsfunktion.*

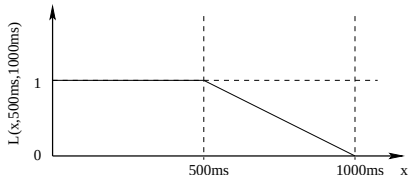


Abbildung 2.3: Reelle Bewertung mittels $L$-Funktion.

Da sich die Bewertung einer Konfiguration aus Bewertungen von Anforderungen zusammensetzt und die Anforderungen an Eigenschaften geknüpft sind, stellen sich zwei Fragen:

- Welche Eigenschaften sollen in die Bewertung einfließen?
- Wie können Werte für diese Eigenschaften bestimmt werden?

Zunächst unterscheidet man zwischen funktionalen und nichtfunktionalen Anforderungen. Obwohl für die Definition der Bewertung nicht vorausgesetzt, werden im Allgemeinen nur nichtfunktionale Anforderungen bewertet. Im weiteren Verlauf wird davon ausgegangen, dass die funktionalen Anforderungen als erfüllt gegeben sind und somit nicht in die Bewertung einfließen. Die Beantwortung der ersten Frage ist zweifelsohne von der betrachteten Domäne abhängig. Eigenschaften die in der einen Domäne von großer Bedeutung sind, können in einer anderen Domäne unwichtig sein. Im Falle von RODOS handelt es sich um die Domäne der eingebetteten Systeme. Auch wenn sich dieses Anwendungsgebiet weiter unterteilen lässt, können einige Eigenschaften von generellem Interesse genannt werden:

- Ausführungszeit
- Kosten
- Energieverbrauch
- Zuverlässigkeit
- Speicherverbrauch
- Gewicht
- geometrische Abmessungen
- Temperaturbereich

Die zweite Frage zielt auf das Aussehen der "Messung", also die Realisierung der Funktion $W(\mathfrak{e})$ ab. Das heißt, für jede Eigenschaft muss ein Weg gefunden werden, aus den Informationen der Wissensbasis einen Wert zu ermitteln.

## 2.2 Zeitverhalten

Für das Zeitverhalten von Systemen gibt es verschiedene Metriken. Man unterscheidet zwischen Zeitverhalten im Extremfall, statistischem Zeitverhalten und durchschnittlichem Zeitverhalten [Malik u. a. 1997]. Unter Zeitverhalten

im Extremfall versteht man typischerweise die WCET (*Worst Case Execution Time*) und in manchen Fällen die BCET (*Best Case Execution Time*). Bei statistischem Zeitverhalten wird davon ausgegangen, dass Deadlines nicht immer eingehalten werden müssen. Stattdessen wird eine Wahrscheinlichkeit für das Einhalten von Deadlines angegeben. Das durchschnittliche Zeitverhalten beschreibt die durchschnittliche Leistung eines Systems, ohne Gewähr Deadlines einzuhalten.

Die Ausführungszeit (engl. Execution Time)[7] ist die Zeit, die ein System benötigt um Eingangsdaten zu verarbeiten und das Ergebnis als Ausgangsdaten bereitzustellen. Sie ist vom System und von den Eingangsdaten abhängig. Weitere Maße für das Zeitverhalten lassen sich bei iterativ arbeitenden Systemen definieren. Bei einem iterativ arbeitendem System werden die Operationen immer wieder auf die jeweils nächsten Eingangsdaten angewendet. Beispielsweise Systeme der Bild- und Signalverarbeitung haben oft diese Eigenschaft. Für solche Systeme ist neben der Latenz der Durchsatz eine wichtige Kenngröße. Der Durchsatz ist die Datenmenge, die ein System pro Zeiteinheit verarbeiten kann. Bei Systemen ohne Fließbandverarbeitung ist er umgekehrt proportional zur Ausführungszeit. Bei Systemen mit Fließbandverarbeitung werden neue Eingangsdaten bereits verarbeitet, bevor das Ergebnis der vorangehenden Eingangsdaten verfügbar ist. Durch Ausnutzung dieser Eigenschaft erhöht sich der Durchsatz.

**Beispiel 2.2.1** *Gegeben sei ein System welches eine Funktion implementiert. Es gibt eine Menge Eingänge* $\mathfrak{X}$*. Bezeichne* $\mathfrak{I}$ *die Menge aller möglichen Belegungen dieser Eingänge, d.h.:*

$$\mathfrak{I} = \{W(\mathfrak{x}_1) \times W(\mathfrak{x}_2) \times \ldots \times W(\mathfrak{x}_{|\mathfrak{X}|})\}, \tag{2.20}$$

*wobei* $W(\mathfrak{x}_i)$ *der Wertebereich des i-ten Einganges ist. Für jede Eingangsbelegung* $\mathfrak{i} \in \mathfrak{I}$ *existiert genau eine zugehörige Ausführungszeit*[8]*. Für eine bestimmte Ordnung der Elemente von* $\mathfrak{I}$ *könnte sich ein Diagramm wie in Abbildung 2.4 ergeben. Die horizontale Achse steht für Eingangsbelegungen,*

[7] Latenz

[8] Schwankungen durch physikalische Einflüsse auf die Implementierung sollen dabei vernachlässigt bleiben.

*die vertikale Achse für die Ausführungszeit. Während es Bereiche gibt, welche durch einfache Abhängigkeit zwischen Eingangsdaten und Ausführungszeit gekennzeichnet sind, gibt es Bereiche mit Unstetigkeiten und wesentlich komplizierteren Abhängigkeiten. Außerdem ist das Aussehen der Kurve nicht zuletzt durch die gewählte Ordnung der Eingangsdaten bestimmt. Bei Eingangsdaten mit mehreren Komponenten, wie Vektoren und Matrizen, stellt sich dann die Frage welche Ordnung zu bevorzugen ist. Abbildung 2.5 zeigt die Abhängigkeit der Iterationsdurchläufe beim Berechnen der Quadratwurzel nach dem Newtonschen Iterationsverfahren von den Eingangsdaten. Das Diagramm hat drei Dimensionen, da die Operation außer vom Radikand noch von einem Parameter Epsilon abhängig ist. Wie zu erwarten, erhöht sich die Anzahl der Durchläufe, je größer der Radikand und je kleiner Epsilon.*

Bei vielen praktischen Anwendungen ist die Menge der Eingangsbelegungen allerdings so groß, dass es unmöglich ist, mit vertretbarem Aufwand für alle Elemente die zugehörige Ausführungszeit zu ermitteln. Mit einer solchen Methode ist es nicht möglich, exakte Aussagen über das Zeitverhalten zu treffen. Um das Problem zu lösen, benötigt man eine formale Charakterisierung der Eingangsdaten und eine Analysemethode, welche diese Charakterisierung nutzen kann. Die formale Charakterisierung von Eingangsdaten ist allerdings ein relativ unerforschtes Gebiet [Malik u. a. 1997]. Es gibt kaum verbreitet angewendete Ansätze einer sinnvollen Spezifikation der Eingangsdaten. Für das Zeitverhalten im Extremfall wird implizit der gesamte kombinierte Wertebereich der Eingangsdaten angenommen. Die Systemumgebung erzeugt aber nicht notwendigerweise alle Kombinationen. Für statistisches Zeitverhalten müsste die Verteilung der Eingangsdaten angegeben werden, wofür keine adäquaten Beschreibungsmittel zur Verfügung stehen. Für das Zeitverhalten im durchschnittlichen Fall können repräsentative Eingangsdaten als Testfälle verwendet werden. Diese zu finden ist oft eine anspruchsvolle, noch weitestgehend nicht automatisierte Aufgabe.

Es existieren jedoch Verfahren zur Abschätzung des Zeitverhaltens im Extremfall. Vor allem bei Systemen mit Echtzeitanforderungen stützt man sich auf die Analyse von WCET und BCET. Im Folgenden wird näher auf

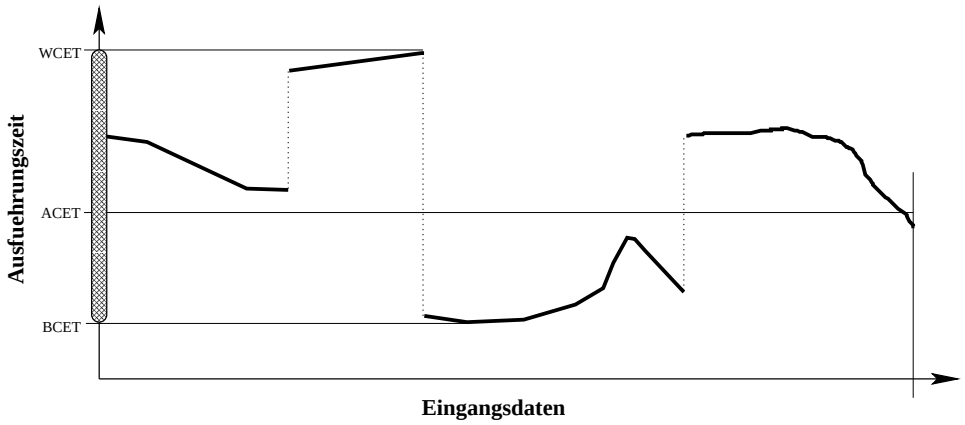


Abbildung 2.4: Ausführungszeit vs. Eingangsdaten

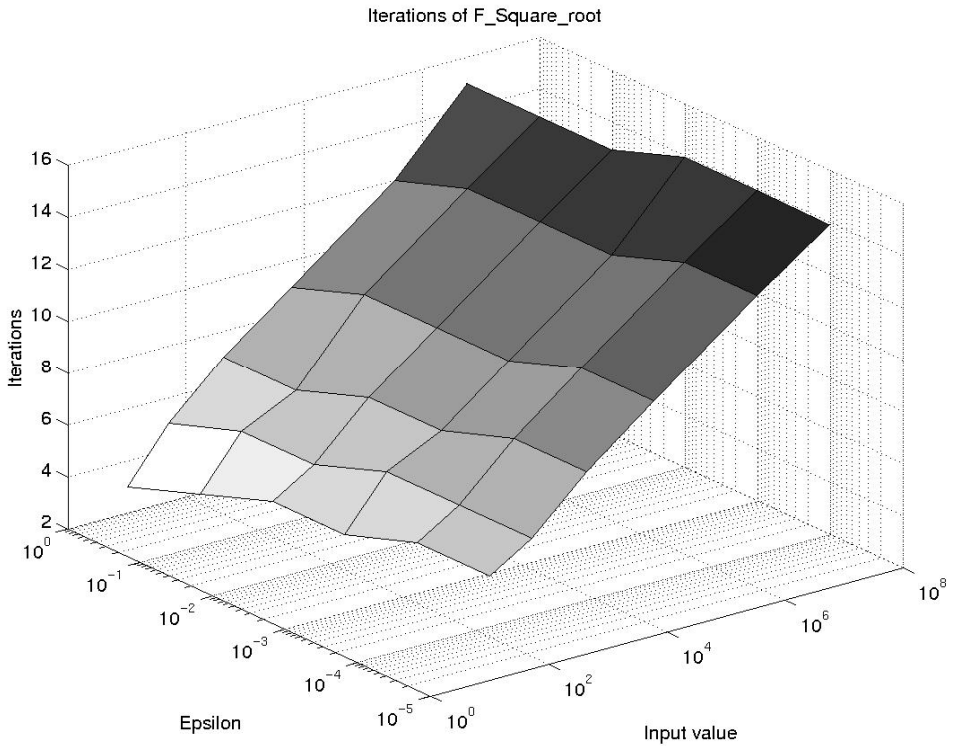


Abbildung 2.5: Iterationsdurchläufe beim Berechnen der Quadratwurzel.

Analyseverfahren der WCET eingegangen. Diese Verfahren ermöglichen die Abschätzung der WCET einer Funktion auf einer bestimmten Ressource. Um das Zeitverhalten eines Systems bestehend aus mehreren Funktionen auf unterschiedlichen Ressourcen zu analysieren, muss eine Ablaufplanung erfolgen. Diese stützt sich auf WCET Abschätzungen der einzelnen Funktionen.

### 2.2.1 Kontroll- und Datenfluss

Um das Zeitverhalten eines Systems analysieren zu können braucht man unter anderem eine Verhaltensbeschreibung des Systems. Zur hierarchischen Modellierung des Kontrollflusses sowie der Datenabhängigkeiten unter den Teilfunktionen eignen sich insbesondere Sequenzgraphen, eine spezielle Klasse des CDFG (*Control Data Flow Graph*). Es folgt eine kurze Darstellung, wie Modulaufrufe (siehe Abb. 2.6) Verzweigung (siehe Abb. 2.7), Iteration (siehe Abb. 2.8) und Datenabhängigkeit in Sequenzgraphen modelliert werden können. Die grau unterlegten Bereiche in den Abbildungen stellen Sequenzgrapheinheiten dar. Die gestrichelten Kanten zwischen ihnen geben den Kontrollfluss wieder. Die Kanten zwischen den Knoten der Einheiten repräsentieren Datenabhängigkeiten.

**Definition 2.2.1 (Sequenzgraph [Teich 1997; Micheli 1994])** *Ein* Sequenzgraph *bezeichnet eine Hierarchie von gerichteten Graphen. Ein generisches Element des Graphen heißt* Einheit eines Sequenzgraphen. *Eine Einheit ist ein erweiterter Datenflussgraph $G = (V, E)$ mit Knotenmenge $V$ und Kantenmenge $E$ sowie folgenden Eigenschaften:*

- *Eine Einheit besitzt zwei Arten von Knoten: a)* Aufgaben *oder Operationen und b)* Hierarchieknoten. *Hierarchieknoten dienen der Verbindung von Einheiten in der Hierarchie.*
- *Eine Einheit stellt einen azyklischen und* polaren *Graphen dar, d.h. es gibt zwei ausgezeichnete Knoten, den sog.* Startknoten *und den* Endknoten. *Beide Knoten sind Hierarchieknoten und stellen die Operation NOP (=keine Operation) dar. Neben Start- und Endknoten gibt es drei weitere Hierarchieknoten, nämlich Modulaufruf (CALL), Verzweigung (BR) und Iteration (LOOP).*

- *Einheiten, die die Blätter der Hierarchie darstellen, besitzen außer dem Start- und Endknoten keine Hierarchieknoten.*

Die Hierarchiebildung erfolgt über Modulaufrufe. Dieses Schema spiegelt die Modellierungsweise der Funktionalität eines Systems in RODOS wieder, indem jeder Klasse der Funktionsschicht, welche Parts besitzt, ein Sequenzgraph zugeordnet wird. Damit lässt sich die Beschreibung des Verhaltens schrittweise mit der Verfeinerung der Funktionen verfeinern. Der *Modulaufrufknoten* (CALL) ist ein Platzhalter für eine andere Einheit des Sequenzgraphen auf niedrigerer Hierarchiestufe. Er modelliert Abhängigkeiten seiner unmittelbaren Vorgängerknoten zum Startknoten des aufgerufenen Moduls und Abhängigkeiten des Endknotens des aufgerufenen Moduls zu den unmittelbaren Nachfolgern des Modulaufrufknotens. Verzweigung wird durch einen *Verzweigungsknoten* (BR) und mehrere Verzweigungsgraphen modelliert. Eine Verzweigung kann als bedingter Modulaufruf aufgefasst werden. Dabei enthält der Verzweigungsknoten einen *Verzweigungsausdruck*, nach dessen Wert einer der Verzweigungsgraphen ausgewählt wird. Die Anzahl unterschiedlicher Verzweigungsgraphen entspricht dem Wertebereich des Verzweigungsausdruckes. Es wird immer genau ein Verzweigungsgraph ausgeführt. In ähnlicher Art und Weise wird Iteration modelliert. Der *Iterationsknoten* (LOOP) zeigt auf den *Iterationsrumpf*, der wiederum durch einen Sequenzgraphen repräsentiert ist. Der Iterationsknoten enthält eine Abbruchbedingung für die Iteration.

Da der Sequenzgraph sowohl Kontroll- als auch Datenfluss beschreibt, gibt es zwei Arten von Kanten: 1) Datenabhängigkeitskanten und 2) Kontrollflusskanten. Die Modellierung von Datenabhängigkeiten erfolgt analog zu einem Datenflussgraph. Besteht eine Abhängigkeit zwischen Funktionen, so wird dies durch eine Datenabhängigkeitskante zwischen den Funktionen dargestellt. Nebenläufigkeit ergibt sich aus den Datenabhängigkeiten, d.h. Funktionen die nicht voneinander abhängen, sind nebenläufig ausführbar, sofern Architektur und Ressourcenbindung es zulassen. Kontrollflusskanten bestehen zwischen den Einheiten des Sequenzgraphen und den verschiedenen Hierarchieknoten.

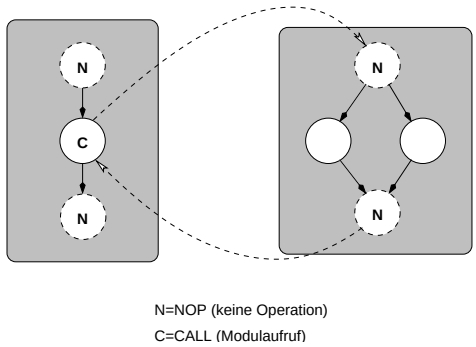


Abbildung 2.6: Sequenzgraph: Modulaufruf. Das Modul selbst ist wiederum eine beliebige Sequenzgrapheinheit.

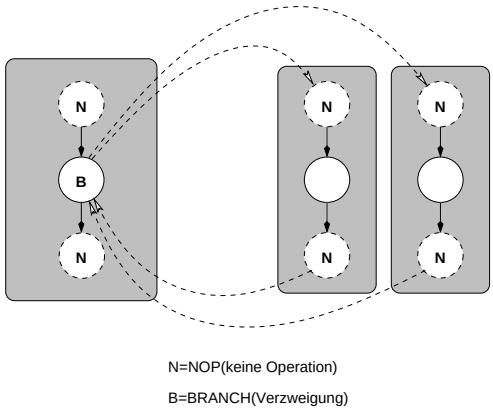


Abbildung 2.7: Sequenzgraph: Verzweigung. Die Auswahl der unterschiedlichen Verzweigungsgraphen geschieht im Verzweigungsknoten.

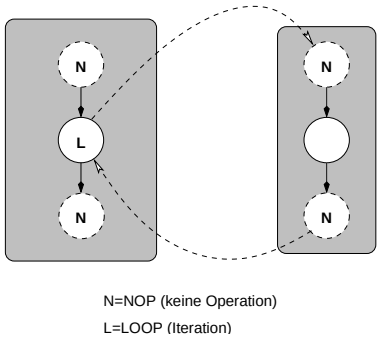


Abbildung 2.8: Sequenzgraph: Iteration. Die Steuerung der Iteration erfolgt im Iterationsknoten.

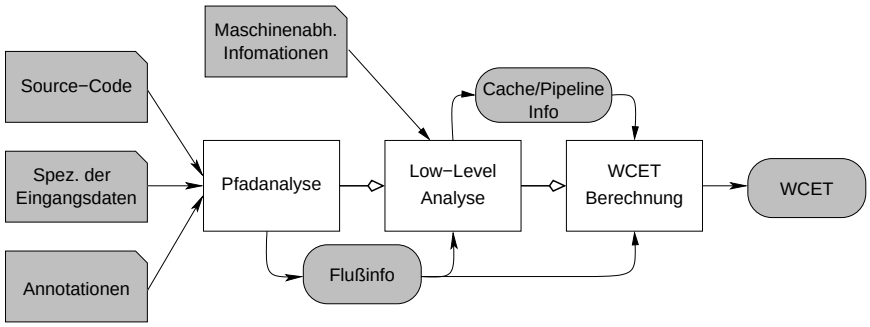


Abbildung 2.9: Übersicht über den Ablauf der WCET-Analyse.

### 2.2.2 Betrachtung von WCET Analyseverfahren

Bei der WCET Analyse wird versucht, eine obere Schranke für die Ausführungszeit einer Funktion auf einer bestimmten Ressource zu ermitteln. Der Prozess der WCET-Analyse kann in verschiedene Phasen unterteilt werden [Engblom u. a. 2000; Malik u. a. 1997]: 1) High-Level-Analyse (auch Pfadanalyse), 2) Low-Level-Analyse und 3) Berechnung der WCET. Abbildung 2.9 zeigt einen verallgemeinerten Ablauf der Zeitanalyse. Konkretere Varianten finden sich in [Lindgren 2000; Engblom u. a. 2000; Vivancos u. a. 2001].

#### Pfadanalyse

Es wird davon ausgegangen die Funktion, deren WCET bestimmt werden soll, sei in Form einer Programmbeschreibung gegeben[9]. Das Ziel von Phase 1 ist es Informationen für mögliche Ausführungspfade durch dieses Programm zu finden. Als Ergebnis liefert die Pfadanalyse, wie oft welche Teilfunktionen gerufen werden, wie oft Schleifen (maximal) durchlaufen werden, welche Abhängigkeiten zwischen verschiedenen Programmzweigen bestehen usw.

**Definition 2.2.2 (Ausführungspfad)** *Sei $G = (V, E)$ der Kontrollflussgraph eines Programmes. Ein* Ausführungspfad *$P$ ist eine Folge von Knoten*

$$P = \{(v_0, v_1, \ldots, v_n) \mid v_i \in V \wedge (v_i, v_{i+1}) \in E\}. \tag{2.21}$$

[9]Die verwendete Sprache ist dabei ohne Bedeutung.

```
for( i=0; i<14; i++ ) {
   for( j= i+1; j<15; j++ ) {
      if ( a[i] < a[j] )
         swap(a[i], a[j]);
   }
}
```

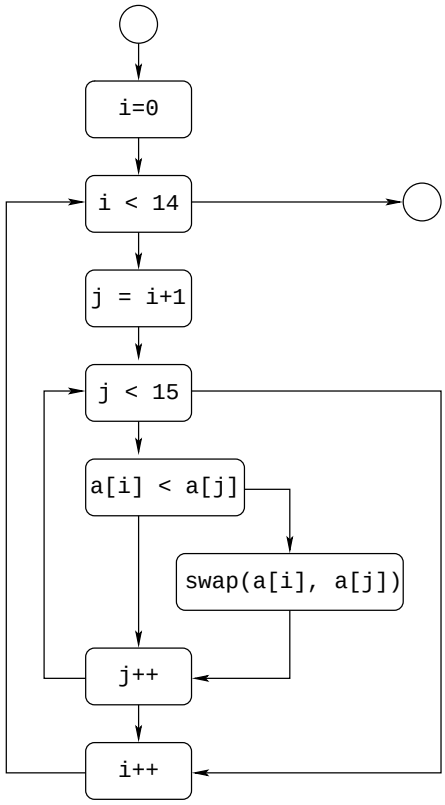


Abbildung 2.10: Funktion und Kontrollflussgraph.

*Jeder Knoten entpricht dabei einer Anweisung. Ein Ausführungspfad kann Schleifen enthalten.*

Abbildung 2.10 zeigt eine Funktionsbeschreibung und ihren Kontrollflussgraph. Die rechteckigen Knoten stehen für Anweisungen, die runden Knoten sind Platzhalter für Start bzw. Ende des Knotrollflusses. Aus obiger Definition folgt die Möglichkeit von unendlich langen Ausführungspfaden. Dies entspricht der Existenz von Endlosschleifen. Um eine sinnvolle Analyse zu gewährleisten wird vorausgesetzt, dass alle Iterationen eine endliche obere Iterationsgrenze besitzen.

Oft sind Abbruchbedingungen von Schleifen oder Verzweigungsausdrücke von den Eingangsdaten abhängig. Die maximale Anzahl von Schleifendurchläufen kann in solchen Fällen nicht durch die Analyse des Quellcodes beschränkt werden. Eine Möglichkeit besteht darin, Testfälle von Eingangsda-

ten zu untersuchen. Die Schwierigkeit besteht darin, Testfälle zu finden, bei denen die Anzahl an Schleifendurchläufen für eine bestimmte Schleife maximal wird. Für komplexe Eingangsdaten lassen sich nur schwer systematisch Testfälle ableiten, so dass häufig nur der Zufallstest bleibt. Ein anderer Ansatz wurde in [Mueller und Wegener 1998] verfolgt. Mit der beschriebenen Methode "Evolutionäres Testen" wird versucht, durch genetische Verfahren geeignete Testfälle zu erzeugen.

In [Engblom und Ermedahl 2000] wird eine Methode beschrieben, Kontrollflussinformationen zu repräsentieren. Es wird gezeigt, wie die Repräsentation benutzt werden kann, um genauere Abschätzungen bezüglich der Ausführungszeiten zu erzielen. Engblom und Ermedahl führen *Scopes* ein, mit denen ein Programm in Blöcke strukturiert wird (Schleifen, Modulaufrufe, etc.). Für die Scopes werden *Scope Facts* definiert, welche Ausführungsanzahlen der Elemente eines Scopes eingrenzen oder zueinender in Beziehung setzten. Beispielsweise kann dies die maximale Anzahl der Durchläufe einer Schleife sein. Die manuelle Annotation ist häufig die einzige praktikable Möglichkeit eine Grenze für die maximale Anzahl von Schleifendurchläufen zu definieren.

### Low-Level-Analyse

Die Low-Level-Analyse versucht die Ausführungszeiten von Basisblöcken eines Programmes auf einer bestimmten Komponente zu ermitteln.

**Definition 2.2.3** *Ein* Basisblock *ist ein Teil eines Kontrollflussgraphen ohne Verzweigungen.*

Dabei können spezielle Architektureigenschaften wie Caches, Pipelines oder Superskalarität berücksichtigt werden. Für jede Komponente, an die eine Funktion gebunden werden kann, muss daher ein eigenes Modell und eine eigene Analysemethode entwickelt werden.

### Berechnungsverfahren der WCET

In der Berechnungsphase werden die Informationen zusammengetragen und eine Abschätzung der WCET berechnet. Im Folgenden werden Methoden zur

Berechnung der WCET betrachtet.

Ein erster Ansatz bei der WCET-Analyse ist, für jede Verzweigung und Schleife des Kontrollflusses anzunehmen, dass immer der Pfad mit der längeren Ausführungszeit genommen wird. Für eine if-then-else Anweisung werden die beiden Teilzweige miteinander verglichen und der zeitlich längere Teilzweig angenommen. Der Nachteil dieses Ansatzes ist, dass Abhängigkeiten zwischen verschiedenen Verzweigungen nicht beachtet werden können. Oft kommt es vor, dass sich die zeitlich längeren Teilzweige unterschiedlicher Verzweigungen im Programm gegenseitig ausschließen. Das führt im Allgemeinen zu einer unnötig großzügigen Abschätzung der WCET. Die Idee ist es, den Programmfluss besser in die Analyse einzubringen.

Eine Funktion sei durch einen Kontrollflussgraph $G = (V, E)$ beschrieben. Die Knoten $v \in V$ repräsentieren Basisblöcke. Für jeden Basisblock existiert eine WCET $c(v)$ und eine maximale Anzahl Ausführungen von $v$ unter allen möglichen Ausführungspfaden durch $G$, bezeichnet mit $\lambda(v)$ . Für einen bestimmten Ausführungspfad $P$ wird jede Kante $e$ des Kontrollflussgraphen $x^P(e)$ mal traversiert. Die gesamte Zeit, die dabei in Basisblock $v$ verbracht wird, ist die Summe der Traversierungen seiner eingehenden Kanten $e \in E_{in}(v)$, multipliziert mit seiner WCET:

$$t(v) = c(v) \sum_{e \in E_{in}(v)} x(e)^P. \tag{2.22}$$

Daraus ergibt sich die gesamte Ausführungszeit des Pfades $P$:

$$\begin{aligned} ET^P &= \sum_{v \in V} t(v) \\ &= \sum_{v \in V} \left( c(v) \sum_{e \in E_{in}(v)} x(e)^P \right). \end{aligned} \tag{2.23}$$

Die Gleichung kann vereinfacht werden, indem $c(v)$ in die innere Summe einbezogen wird. Die geschachtelte Summe iteriert über alle Eingangskanten aller Knoten des Kontrollflussgraphen. Da keine Kontrollflusskanten ohne Zielknoten existieren, d.h. jede Kontrollflusskante stets eingehende Kante

eines Knotens ist, kann äquivalent über alle Kanten summiert werden:

$$ET^P = \sum_{e \in E} x(e)^P c(v_T(e)), \tag{2.24}$$

wobei $v_T(e)$ den Zielknoten der Kante $e$ bezeichnet.

Um die WCET des Programmes zu finden, muss der maximale Wert für $ET^P$ für alle möglichen Ausführungspfade gefunden werden. Sei $\chi^G$ die Menge aller Ausführungspfade eines Kontrollflussgraphen $G = (V, E)$, dann

$$WCET = \max_{P \in \chi^G} ET^P = \max_{P \in \chi^G} \sum_{e \in E} x(e)^P c(v_T(e)). \tag{2.25}$$

Eine einfache Möglichkeit, eine Lösung für Gleichung 2.25 zu bestimmen ist es, Testfälle für alle $P$ anzulegen. Weiter oben wurde bereits erwähnt, dass die Menge der Eingangsdaten $\mathfrak{I}$ sehr groß sein kann. Zwar ist die Anzahl der Ausführungspfade $P$ begrenzt, da $\| \chi^G \| \leq \| \mathfrak{I} \|$, doch es muss davon ausgegangen werden, dass nicht alle Pfade untersucht werden können. Es würde aber genügen, wenn sich die Eingangsbelegung, die zur wirklichen WCET führt, unter den Testfällen befindet. Hierzu bietet die bereits erwähnte Methode des "Evolutionären Testens" einen Ansatz.

Eine andere Möglichkeit besteht darin, die Anzahl der zu betrachtenden Pfade zu beschränken, indem die Abhängigkeiten zwischen ihnen berücksichtigt werden. In [Park 1992] werden ausführbare Pfade durch reguläre Ausdrücke beschrieben. Der Anwender kann Pfadinformationen über die Scriptsprache IDL (*Information Description Language*) zur Verfügung stellen. Über Schnittmengen von regulären Ausdrücken werden dann ausführbare Pfade ermittelt und nur diese untersucht.

Alternative Methoden lösen das Maximierungsproblem in Gleichung 2.25 mittels ILP (*Integer Linear Programming*) [Puschner und Koza 1995; Li und Malik 1995]. Offensichtlich können die $x(e)$ keine beliebigen Werte annehmen. Zum Beispiel sind alle $x(e)$ nichtnegative ganze Zahlen und ihr maximaler Wert ist durch $\lambda(v_S(e))$ gegeben. also

$$x(e) \in \mathbb{N}_0 \ \wedge \ x(e) \leq \lambda(v_S(e)). \tag{2.26}$$

Außerdem können die $x(e)$ Abhängigkeiten untereinander aufweisen. Diese Abhängigkeiten werden durch lineare Restriktionen[10] zwischen verschiedenen $x(e)$s beschrieben, so dass die $x(e)$ nur Werte annehmen können, welche Ausführungspfade repräsentieren [Puschner und Koza 1995; Li und Malik 1995]. Die Maximierung kann damit durch ILP erfolgen.

Folgende fundamentale Restriktionen beschreiben die Erhaltung des Flusses. Für jeden Knoten $v \in V$ gilt, dass die Summe der Traversierungen der eingehenden Kanten gleich der Summe der Traversierungen der ausgehenden Kanten ist, d.h.

$$\sum_{e_i \in E_{in}(v)} x(e_i) = \sum_{e_o \in E_{out}(v)} x(e_o) \quad \forall v \in V. \tag{2.27}$$

Für Schleifen müssen zusätzliche Restriktionen definiert werden. Mit den obigen Restriktionen ist das Szenario denkbar, in dem die eingehende Kante eines Schleifenkopfes nie traversiert wird, aber den Kanten innerhalb der Schleife Werte größer Null zugeordnet werden können. Die Tatsache, dass eine Schleife nur dann ausgeführt wird, wenn der Kontrollfluss zum Schleifenkopf gelangt, muss explizit durch Restriktionen formuliert werden:

$$\sum_{e_i \in E_{in}(v) - E_{back}(v)} x(e_i) \sum_{e_b \in E_{back}(v)} \lambda(v_S(e_b)) \geq \sum_{e_b \in E_{back}(v)} x(e_b). \tag{2.28}$$

Außerdem können noch beliebige andere Restriktionen hinzugefügt werden, welche Abhängigkeiten zwischen Programmteilen zum Ausdruck bringen. Solche Restriktionen können zum Beispiel das Ergebnis einer Pfadanalyse sein.

### 2.2.3 BCET Analyse

Analog zur Berechnung der WCET mit ILP kann auch die BCET ermittelt werden. Dazu müssen für alle Schleifen untere Schranken für die Anzahl der Durchläufe existieren. Es kann ein Minimierungsproblem mit entsprechenden Restriktionen formuliert werden, welches mittels ILP einen Wert für die BCET liefert.

[10] auch Constraints genannt

### 2.2.4 Ablaufplanung

Die Verteilung der Funktionen eines Systems auf seine Ressourcen hat Einfluss auf dessen Zeitverhalten. Solange an jede Ressource nur eine Funktion gebunden ist, ist die nebenläufige Ausführung aller Funktionen, zwischen denen keine Datenabhängigkeiten existieren, immer möglich. Geteilte Ressourcen kommen immer dann zustande, wenn mehrere Funktionen an ein und die selbe Ressource gebunden sind.

**Definition 2.2.4 (Bindung)** *Gegeben sei eine Menge von Funktionen $V$ und die Menge der Ressourcen einer Architektur $V_R$. Unter* Bindung *versteht man die Zuordnung von Funktionen zu Ressourcen, d.h. eine Funktion*

$$\beta : V \mapsto V_R, \tag{2.29}$$

*wobei $v_r = \beta(v)$ die Ressource repräsentiert, an welche die Funktion $v$ gebunden ist.*

Das Ziel der Ablaufplanung besteht darin, eine zeitliche Abfolge der Ausführung von Funktionen zu bestimmen, so dass keine Datenabhängigkeiten verletzt werden und zu keinem Zeitpunkt mehrere Funktionen auf der gleichen Ressource ausgeführt werden.

Die Ablaufplanung kann in zwei unterschiedliche Kategorien unterteilt werden: a) statische Ablaufplanung und b) dynamische Ablaufplanung. Die dynamischen Ablaufplanung findet erst zur Laufzeit, also während des Betriebs eines Systems statt. Dies ist bei Systemen notwendig, bei denen zur Entwurfszeit das Ablaufplanungsproblem nicht vollständig bekannt ist. Dynamische Ablaufplanung kommt beispielsweise bei Multitaskingsystemen, die mehrere Aufgaben im Zeitmultiplex abarbeiten, und zur Laufzeit entscheiden, wann welche Aufgabe als nächste eine Ressource beanspruchen darf, zur Anwendung. Im Gegensatz dazu erfolgt die statische Ablaufplanung zur Entwurfszeit. Alle Aufgaben, d.h. Funktionen, die das System abarbeiten muss sind im voraus bekannt. Ausserdem sind Werte für die WCET aller Funktionen bekannt. Besonders im Bereich der eingebetteten Systeme kann von diesen Voraussetzungen ausgegangen werden, da die Systemumgebung meist

wohldefiniert ist. Aus diesen Gründen soll die statische Ablaufplanung die Grundlage der zeitlichen Analyse bilden.

Für die Lösung von statischen Ablaufplanungsproblemen existieren eine Reihe von Algorithmen. Verfeinerte ASAP (*As Soon As Possible*) [Tseng und Siewiorek 1986] und ALAP (*As Late As Possible*) Algorithmen, Listscheduling oder Force-Directed-Scheduling [Paulin und Knight 1989] erlauben die Planung unter Berücksichtigung von Ressourcenbeschränkungen. Eine weitere Möglichkeit ist die Abbildung des Ablaufplanungsproblems auf ein ganzzahliges lineares Programm (ILP) [Teich 1997, S.96-114].

Es sei darauf hingewiesen, das die latenzoptimale Ablaufplanung mit Ressourcenbeschränkung NP-hart ist [Coffman 1976]. D.h. es existiert kein effizienter Algorithmus zur Lösung des Ablaufplanungsproblems mit minimaler Latenz. Die obengenannte Möglichkeit der Abbildung auf ein ganzzahliges lineares Programm ist ebenfalls NP-hart. Die anderen genannten Algorithmen liefern im Allgemeinen suboptimale Ergebnisse, beanspruchen allerdings nur polynomielle Laufzeit. Da die Ablaufplanung im Rahmen der Bewertung von Konfigurationen während des Konfigurationsalgorithmus ausgeführt werden muss, und weiter oben bereits erwähnt wurde, dass die Exaktheit der Lösung nicht von primärer Bedeutung ist, soll ein erweiterter ASAP-Algorithmus zur Ablaufplanung eingesetzt werden.

**Definition 2.2.5 (Ablaufplan [Teich 1997, S.84,85](sinngem.))** *Ein* Ablaufplan *einer Sequenzgrapheinheit $S = (V, E)$ ist eine Funktion $\tau : V \mapsto \mathbb{Z}_0^+$, die jeder Funktion $v_i \in V$ die Startzeit $t_i = \tau(v_i)$ zuordnet, so dass gilt:*

$$\tau(v_i) \geq \tau(v_j) + d_j \qquad \forall (v_i, v_j) \in E, \tag{2.30}$$

*wobei $d_i$ die Ausführungszeit für $v_i$ darstellt.*

**Definition 2.2.6 (Latenz [Teich 1997, S.85](sinngem.))** *Die Latenz $L$ eines Ablaufplans $\tau$ einer Sequenzgrapheinheit $G = (V, E)$ ist definiert als*

$$L = \max_{v_i \in V}\{\tau(v_i) + d_i\} - \min_{v_i \in V}\{\tau(v_i)\} \tag{2.31}$$

*und bezeichnet damit die Länge des Zeitintervalls vom Startzeitpunkt der ersten Funktion bis zum Ende der Ausführung der letzten Funktion.*

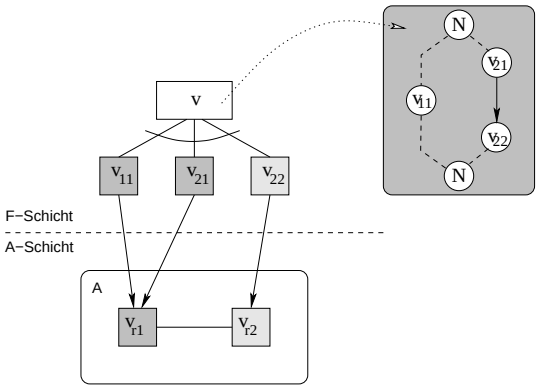


Abbildung 2.11: Geteilte Ressourcen: Bindung und CDFG.

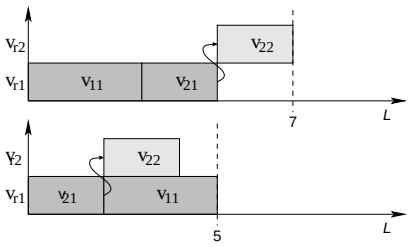


Abbildung 2.12: Geteilte Ressourcen: Ablaufpläne.

Es wird angenommen, dass die Ausführungszeiten und Bindungen aller Funktionen bekannt sind. Diese Annahme erweist sich als realistisch unter Verweis auf die WCET-Analyse (s. Abschnitt 2.2.1), deren Aufgabe die Bestimmung der Ausführungszeiten von Funktionen auf bestimmten Ressourcen ist. In Abbildung 2.11 ist die Beschreibung der Funktionen und deren Bindung an Ressourcen einer Architektur im Schichtenmodell von RODOS zu sehen. Die Funktion $v$ ist unterteilt in drei Teilfunktionen. Die Datenabhängigkeiten sind im Sequenzgraph oben rechts beschrieben. Während $v_{11}$ keine Datenabhängigkeiten zu den anderen beiden Funktionen besitzt, ist die Funktion $v_{22}$ von den Daten der Funktion $v_{21}$ abhängig. Die Relationen zwischen Funktionsschicht und Architekturschicht beschreiben die Bindung der Funktionen an die Ressourcen der Architektur $A$. Im Beispiel sind die Funktionen $v_{11}$ und $v_{21}$ an die Ressource $v_{r1}$ gebunden. $v_{r1}$ stellt damit eine geteilte Ressource dar. Die Funktion $v_{22}$ ist an die Ressource $v_{r2}$ gebunden. Abbildung 2.12 zeigt Beispiele für Ablaufpläne mit unterschiedlicher Latenz $L$ für das Szenario in

Abbildung 2.11. Bei dem oberen Ablaufplan werden als erstes die Funktionen $v_{11}$ und $v_{21}$ ausgeführt. Aufgrund der Datenabhängigkeiten zwischen $v_{21}$ und $v_{22}$ kann $v_{22}$ erst zum Zeitpunkt 5 ausgeführt werden. Vertauscht man die Reihenfolge von $v_{11}$ und $v_{21}$, so ergibt sich der untere Ablaufplan, welcher eine kürzere Latenz hat. Die Funktion $v_{22}$ kann nebenläufig ausgeführt werden.

Algorithmus 1 zeigt den entwickelten Ablaufplanungsalgorithmus SCHEDULE in vereinfachter Form. Die Eingangsparameter stellen eine Sequenzgrapeinheit $\mathcal{SG}$, die zu planende Funktion currFunc[11] und die aktuelle Planungszeit currTime dar. Er liefert als Ergebnis ein Zeitintervall $[von, bis]$, welches die Ausführungszeit der Funktion currFunc repräsentiert. Als Vorbedingung für den Algorithmus müssen die Startzeiten aller Funktionen gleich Null sein.

Der Algorithmus kann in zwei unterschiedliche Teile zerlegt werden, in welche von Zeile 3 aus verzweigt wird. Ist die zu planende Funktion keine Basisfunktion, wird der erste Teil (Zeilen 4-31) ausgeführt. Dieser Teil plant alle Funktionen in der Sequenzgrapeinheit $\mathcal{SG}$. Dazu wird über alle Teilfunktionen in der Reihenfolge ihrer topologischen Sortierung iteriert. Diese Reihenfolge stellt sicher, dass die Ablaufplanung alle Datenabhängigkeiten erfüllt.

Im ersten Abschnitt (Zeilen 6-23) der Iteration wird für jede Teilfunktion func SCHEDULE mit der Planungszeit gleich der Startzeit der Teilfunktion rekursiv aufgerufen (Zeile 9). Das entspricht einem Abstieg in der Funktionshierarchie, wobei die Teilfunktionen der Teilfunktionen usw. geplant werden. Anschließend wird die Endzeit der aktuellen Funktion ensprechend der Planungsresultate der Teilfunktionen erhöht (Zeile 10). Dann wird für die Teilfunktion func geprüft, ob Ressourcenkonflikte mit anderen Funktionen, welche an die gleiche Ressource wie func gebunden sind, existieren. Ein Ressourcenkonflikt kommt zustande, wenn folgende Bedingung erfüllt ist:

$$\text{f.startTime} < \text{func.endTime} \wedge \text{f.endTime} > \text{func.startTime} \tag{2.32}$$

[11] currFunc ist ein Knoten der Sequenzgrapheinheit $\mathcal{SG}$.

**Algorithmus 1** Hierarchische ASAP Ablaufplanung mit Ressourcenbeschränkung.

```
 1: [von, bis] SCHEDULE(SG,currFunc,currTime)
 2: maxTime := currTime
 3: if not Basisfunktion(currFunc) then
 4:   TOPSORT(SG)
 5:   for all func ∈ SG do
 6:     repeat
 7:       func.startTime := max(func.startTime, currTime)
 8:       for all subSG ∈ func.SUBSG do
 9:         [v, b] := SCHEDULE(subSG,func,func.startTime)
10:         maxTime := max(maxTime, b)
11:       end for
12:       for all res ∈ {von func benötigte Ressourcen} do
13:         for all f ∈ {an res gebundene Funktionen} do
14:           if Ressourcenkonflikt(f,func) then
15:             if f noch nicht geplant then
16:               f.startTime := max(f.startTime, maxTime)
17:             else
18:               func.startTime := f.endTime
19:             end if
20:           end if
21:         end for
22:       end for
23:     until keine Ressourcenkonflikte mehr
24:     for all f ∈ {von func abhängige Funktionen} do
25:       f.startTime := max(f.startTime, maxTime)
26:     end for
27:     for all f ∈ {noch nicht gebundene Funktionen} do
28:       if f noch nicht geplant then
29:         f.startTime := max(f.startTime, maxTime)
30:       end if
31:     end for
32:   end for
33: else
34:   maxTime := currTime + WCET_Analyse(currFunc)
35: end if
36: func.endTime := maxTime
37: return [func.startTime, func.endTime]
```

Ist ein Ressourcenkonflikt vorhanden und die den Konflikt verursachende Funktion wurde bereits geplant, so muss die aktuelle Teilfunktion func mit neuer Startzeit geplant werden (Zeile 18)[12]. Wurde die verursachende Funktion noch nicht geplant, so kann deren Startzeit auf die Endzeit von func verschoben werden, womit der Ressourcenkonflikt beseitigt wird. Diese Behandlung von Ressourcenkonflikten führt auch zur Suboptimalität des Algorithmus. Es kann nicht entschieden werden, ob die Verschiebung von func oder der verursachenden Teilfunktion bessere Ergebnisse erzielt. An dieser Stelle müsste ein optimaler Algorithmus verzweigen, beide Möglichkeiten verfolgen, vergleichen und die bessere Variante wählen. Da die Situation prinzipiell bei jeder Teilfunktion möglich ist, können so viele Verzweigungen wie Teilfunktionen existieren vorkommen. Das bedeutet, ein optimaler Algorithmus muss $2^{\text{Anzahl Teilfkt.}}$ Varianten untersuchen, was exponentielle Lauftzeit zur Folge hat.

Im zweiten Abschnitt (Zeilen 24-26) der Iteration werden alle von func abhängigen Teilfunktionen verschoben, um die Datenabhängigkeiten zu erfüllen. Im dritten Abschnitt (Zeilen 27-31) der Iteration werden alle Teilfunktionen, welche noch nicht gebunden sind, auf den Endzeitpunkt von func verschoben. Diese Maßnahme beruht auf der Annahme, dass die Bindung der Funktionen potentiell zu Ressourcenkonflikten führen kann.

Wenn die zu planende Funktion currFunc eine Basisfunktion ist, ermittelt SCHEDULE im 2. Teil (Zeile 33) die WCET für diese Funktion, unter Berücksichtigung ihrer Bindung (s. Zeile 33). Ist currFunc an eine bestimmte Ressource gebunden, so wird die WCET der Funktion auf dieser Ressource benutzt. Ist currFunc noch nicht gebunden[13], so wird die maximale WCET unter allen möglichen Bindungen von currFunc benutzt.

Der Algorithmus SCHEDULE erzeugt einen Ablaufplan, der allen Funktionen unter Berücksichtigung von Datenabhängigkeiten und Ressourcenbeschränkungen Start- und Endzeiten zuordnet. Alle Aussagen über das Zeitverhalten eines Systems werden anschließend auf einen solchen Ablaufplan

[12]Diese Verschiebung der Startzeit führt noch nicht zur Beseitigung des Ressourcenkonfliktes, weshalb die until-Bedingung in Zeile 23 noch nicht erfüllt ist. Dadurch kommt es zur erneuten Planung in Zeile 9.

[13]Dies kann in einer zu bewertenden unvollständigen Konfiguration vorkommen.

zurückgeführt.

Der Algorithmus unterscheidet zwischen den Knotentypen "CDFG_CALL", "CDFG_LOOP" und "CDFG_BRANCH". Bei der Planung eines Schleifenknotens "CDFG_LOOP" wird die ermittelte Latenz mit der maximalen Anzahl von Schleifendurchläufen multipliziert. Diese kann in der Sequenzgraphbeschreibung definiert werden. Im Falle eines Verzweigungsknotens wird die Latenz des zeitlich längsten Teilzweiges verwendet.

Weiterhin kann der Algorithmus, in leicht modifizierter Form, einen Ablaufplan mit einer unteren, bzw. oberen Latenzschranke erstellen. Der Unterschied der beiden Varianten liegt nur im zweiten Teil des beschriebenen Algorithmus. Während der Algorithmus für die oberer Latenzschranke immer die maximal mögliche WCET jeder Basisfunktion benutzt, verwendet er die minimal mögliche WCET der Basisfunktionen, um einen Ablaufplan mit der unteren Latenzschranke zu erstellen. Die Differenz der Latenz der beiden Ablaufpläne stellt die mögliche Schwankungsbreite der Latenz, bzw. WCET des Gesamtsystems dar. Wenn alle Basisfunktionen an eine Ressource gebunden sind, ergibt sich in beiden Fällen der gleiche Ablaufplan und die Schwankungsbreite der WCET ist gleich null.

### 2.2.5 Weitere zeitliche Kenngrößen

Aus dem Ablaufplan können einige zeitliche Kenngrößen ermittlt werden. Für jede Ressource kann der Grad der Ausnutzung definiert werden:

**Definition 2.2.7 (Grad der Ausnutzung)** *Gegeben seien die Menge der Funktionen $V$, die Menge der Ressourcen $V_R$, eine Bindung $\beta$ sowie ein Ablaufplan $\tau$. Der Grad der Ausnutzung der Ressource $v_r \in V_R$ ist gegeben durch*

$$U(v_r) = \frac{100}{L_\tau} \sum_{v_i \in \beta^{-1}(v_r)} d_i, \tag{2.33}$$

*wobei $L_\tau$ die Latenz des Ablaufplanes $\tau$ und $d_i$ die Ausführungszeit der Funktion $v_i$ repräsentiert.*

Der Grad der Ausnutzung ist damit das Verhältnis der Zeit, die eine Ressource aktiv ist, zur Latenz des Systems.

**Definition 2.2.8** *Der Anteil der Ausführungszeit einer jeden Funktion $v \in V$ an der Gesamtausführungszeit ist gegeben durch:*

$$u(v) = \frac{d_i}{L_\tau} \cdot 100. \tag{2.34}$$

**Definition 2.2.9 (Verarbeitungsrate)** *Die Verarbeitungsrate ist der Kehrwert der Latenz*[14]*:*

$$R = \frac{1}{L_\tau}. \tag{2.35}$$

## 2.3 Kosten

Die Kosten einer Konfiguration sind von der verwendeten Architektur abhängig. Anzahl und Typ der verwendeten Ressourcen entscheiden maßgeblich über die Gesamtkosten.

**Definition 2.3.1 (Kosten)** *Gegeben sei die Menge der Ressourcen $V_R$. Die Funktion $c_b : V_R \mapsto \mathbb{Z}_0^+$ beschreibt die* Basiskosten *$c_b(v_r)$ der Ressource $v_r \in V_R$.*

### 2.3.1 Einfaches Kostenmodell

Im einfachen Kostenmodell werden nur die Kosten der Ressourcen modelliert. Eine an eine Ressource gebundene Funktion verursacht keine zusätzlichen Kosten für diese Ressource. Die Kosten einer Konfiguration ergeben sich somit aus der Summe der Basiskosten der Komponenten der Architektur:

$$c = \sum_{v \in V_A} c_b(v). \tag{2.36}$$

### 2.3.2 Kostenmodell für geteilte Ressourcen

In einem detaillierteren Kostenmodell [Teich 1997, S.383] kommen zu den Basiskosten einer Ressource zusätzliche Kosten für an sie gebundene Funktionen hinzu.

[14] Gilt nur für Systeme ohen funktionale Fließbandverarbeitung.

**Definition 2.3.2 (Addcost)** *Gegeben sei eine Bindung $\beta$. Die Menge $E_B = (v, v_r)|v_r = \beta(v)$ ist die Menge aller Kanten von Funktionen zu Ressourcen. Die Funktion $c_a : E_B \mapsto \mathbb{Z}_0^+$ beschreibt für jede Funktion $v \in V$ entstehende* Zusatzkosten *bei Bindung auf die Ressource $v_r \in V_R$.*

Mit den Zusatzkosten können zum Beispiel Kosten für Programmspeicher oder Chipfläche ausgedrückt werden. Für Situationen, bei denen sich verschiedene Aufgaben den gleichen Programmspeicher oder das gleiche Hardwaremodul teilen ist die Annahme additiver Zusatzkosten falsch. Dazu werden Funktionstypen definiert, welche bei der Kostenberechnung entsprechend behandelt werden können. So verursachen alle an eine Ressource gebundene Aufgaben des gleichen Funktionstyps nur einmal Zusatzkosten:

**Definition 2.3.3 (Funktionstypen)** *Die Menge der Aufgaben $V$ ist partitioniert in* Typen *$T_i \in \mathbf{T}$, d.h. jede Aufgabe gehört genau zu einem Typ. Alle Aufgaben gleichen Typs $T_i$, die an die selbe Ressource gebunden sind, verursachen lediglich die zusätzlichen Kosten*

$$c_t(T_i, v_r) = \max\{c_a(e_B)|e_B = (v, v_r) \in E_B \wedge v \in T_i\} \tag{2.37}$$

Mit obigen Definitionen ergeben sich die Kosten einer Konfiguration aus der Summe der Kosten für die vorkommenden Funktionstypen und der Basiskosten der Komponenten:

$$c = \sum_{v_r \in V_R} \left( c_b(v_R) + \sum_{T_i \in \mathbf{T}} c_t(T_i, v_r) \right). \tag{2.38}$$

Mit diesem Kostenmodell können nicht nur Kosten im eigentlichen Sinne modelliert werden, sondern alle additiven Eigenschaften.

**Definition 2.3.4 (Additive Eigenschaften)** *Sei $v$ eine Klasseninstanz mit mehreren Parts. Als* additive Eigenschaften *von $v$ werden die Eigenschaften bezeichnet, welche sich durch Addition aus den entsprechenden Eigenschaften der Parts von $v$ berechnen lassen.*

Weitere additive Eigenschaften sind zum Beispiel Gewicht oder Energieverbrauch. Im Fall von Energieverbrauch haben alle Komponenten einen Gundenergiebedarf. Jede Funktion erhöht aber evtl. den Energiebedarf, was mit Zusatzkosten modelliert werden kann.

## 2.4 Sonstige Eigenschaften

Außer den zeitlichen und additiven Eigenschaften gibt es beliebige andere Eigenschaften. Die Eigenschaft "Temperaturbereich" könnte beispielsweise durch den umschließenden Intervall aller Temperaturbereiche der Parts einer Klasseninstanz definiert werden. Dies führt zu einer Klasse Eigenschaften, welche als Intervalleigenschaften bezeichnet werden können. Ein Maßfür die Zuverlässigkeit könnte durch das Produkt der Zuverlässigkeiten der Parts definiert sein. Solche Eigenschaften könnten als multiplikative Eigenschaften bezeichnet werden. Prinzipiell sind für jede Art von Eigenschaften geeignete Attributmethoden denkbar.

# Kapitel 3

# Analyse und Bewertung von Konfigurationen in RODOS

## 3.1 Repräsentation von nichtfunktionalen Anforderungen

Alle nichtfunktionalen Anforderungen werden als Attribute in der Topklasse der Spezifikationsschicht definiert durch Angabe von:

- Name
- Minimal zulässigem Wert
- Maximal zulässigem Wert.

Minimaler und maximaler Wert bilden das *Anforderungsintervall*. Die Attribute heißen *Anforderungsattribute*. Diese Beschreibung von Anforderungen beinhaltet folgende Bedeutung:

- Die Anforderung richtet sich an die Eigenschaft der Konfiguration mit dem gleichen Namen.
- Die Anforderung gilt als erfüllt, wenn die korrespondierende Eigenschaft die Grenzen des Anforderungsintervalls nicht überschreitet.

Im Sinne von Definition 2.1.2 ist jede Anforderung $\mathfrak{a}(x)$, repräsentiert durch das Anforderungsattribut $a$, eine Aussageform:

$$\mathfrak{a}(x) : x \ <\text{op}> \ [a_{min}, a_{max}], \tag{3.1}$$

wobei <op> für einen Operator steht, der durch die Bewertungsfunktion realisiert wird.

**Beispiel 3.1.1** *Es sollen drei Anforderungen an eine Konfiguration gestellt werden. Die Ausführungszeit soll im Intervall* $[10, 20]$ *liegen. Die Kosten sollen in* $[15, 45]$ *und der Energieverbrauch soll in* $[9, 17]$ *liegen. Dazu werden in der Topklasse der Spezifikationsschicht folgende Attribute vereinbart:*

```
attribute (
  WCET:    generic real from 10 to 20;
  COST:    generic real from 15 to 45;
  PWR:     generic real from 9 to 17
);
```

## 3.2 Repräsentation von Eigenschaften

Die Eigenschaften einer Konfiguration werden durch Attribute in der Topklasse der Funktionsschicht repräsentiert. Sie sind definiert durch:

- Deklaration von *analysis*
- Name
- minimaler Wert
- maximaler Wert
- Attributmethode.

Die Deklaration von *analysis* dient zur Markierung des Attributes als Eigenschaft. Der Name des Attributes dient dazu, die Verbindung zwischen Eigenschaft und Anforderung herzustellen. Für jede Anforderung muss eine Eigenschaft mit gleichem Namen existieren. Minimum und Maximum definieren das *Eigenschaftsintervall*. Die Attribute werden als *Eigenschaftsattribute* bezeichnet. Die angegebene Attributmethode bestimmt die Grenzen des Eigenschaftsintervalls. Sie realisiert die in Definition 2.1.1 benutzte Funktion $M(\mathfrak{e})$.

**Beispiel 3.2.1** *Für die weiter oben definierten Anforderungen sollen geeignete Eigenschaften der Konfiguration hinzugefügt werden. Dazu werden in der Topklasse der Funktionsschicht folgende Attribute vereinbart:*

```
attribute (
  WCET:    analysis real from 0 to infinity methods(WCET);
  COST:    analysis real from 0 to infinity methods(COST);
  PWR:     analysis real from 0 to infinity methods(COST)
);
```

## 3.3 Bewertung der Anforderungen

Für die Bewertung der Anforderungen in Bezug auf die Eigenschaften wurden verschiedene Bewertungsfunktionen eingeführt. Laut Definition 2.1.3 muss die Bewertungsfunktion der Anforderung, in Abhängigkeit des Wertes der entsprechenden Eigenschaft, einen Wert in $[0, 1]$ zuordnen. Die Eigenschaften einer Konfiguration in RODOS sind keine Einzelwerte, sondern Intervalle. Daher setzen die eingeführten Bewertungsfunktionen Anforderungsintervall und Eigenschaftsintervall in Beziehung und liefern einen Wert zwischen 0 und 1. Tabelle 3.1 zeigt eine Übersicht über die Bewertungsfunktionen. Die Algorithmen der Bewertungsfunktionen befinden sich im Anhang in Abschnitt A.3.

### 3.3.1 Interval match

Die Funktion `interval_match` stellt eine binäre Bewertungsfunktion dar. Die zu bewertende Anforderung und ihre entsprechende Eigenschaft werden der Funktion als Parameter übergeben. Die Funktion liefert den Wert 1, falls das Eigenschaftsintervall eine Teilmenge des Anforderungsintervalls ist, sonst liefert sie den Wert 0.

### 3.3.2 Interval overlap

Die Funktion `interval_overlap` stellt eine reelle Bewertungsfunktion dar. Sie liefert den Wert 1, falls das Eigenschaftsintervall eine Teilmenge des An-

| Bewertungsfunktion | Wertebereich | Beschreibung |
|---|---|---|
| `interval_match` | $\{0, 1\}$ | Liefert 1 wenn das Eigenschaftsintervall Teilmenge des Anforderungsintervalls ist, sonst 0. |
| `interval_overlap` | $[0, 1]$ | Liefert das Verhältnis des überlappenden Teils beider Intervalle zur Länge des Eigenschaftsintervalls. |
| `interval_distance` | $[0, 1]$ | Liefert Werte in $[0.5, 1]$, wenn sich die Intervalle überlappen, sonst Werte in $[0, 0.5]$, umgekehrt proportional zum Abstand der Intervalle. |

Tabelle 3.1: Übersicht der Bewertungsfunktionen.

forderungsintervalls ist. Liegt das Eigenschaftsintervall teilweise außerhalb des Anforderungsintervalls, so ergibt sich ein Wert kleiner 1, der dem Verhältnis des überlappenden Teils beider Intervalle zur Länge des Eigenschaftsintervalls entspricht. Überlappen sich die Intervalle nicht, so liefert die Funktion den Wert 0. Die Folge dieser Bewertung ist, dass Konfigurationen, welche die Anforderung nicht notwendigerweise erfüllen, einen Wert größer 0 erhalten. Für den Konfigurierer hat das den Vorteil, dass er solche Konfigurationen noch unterscheiden kann und ein Kriterium für die weitere Suche hat. Der Nachteil dieser Bewertung kommt bei kurzen Eigenschaftsintervallen zum Tragen. Je kürzer das Intervall, desto schneller geht die Bewertung gegen 0.

### 3.3.3 Interval distance

Die Funktion `interval_distance` stellt eine reelle Bewertungsfunktion dar. Sie liefert 1, wenn sich beide Intervalle überlappen. Anderenfalls liefert die Funktion einen Wert in $[0, 1]$, der umgekehrt proportional zum Abstand der Intervalle zueinander ist. Der Abstand ist dabei die "Lücke" zwischen den Intervallgrenzen der beiden Intervalle. Diese Bewertung hat gegenüber `interval_overlap` den Vorteil, das der Konfigurierer auch außerhalb der Anforderungen liegende Konfigurationen in ihrer Bewertung unterscheiden

kann. Allerdings bedeutet hier eine Bewertung kleiner 1, dass von dieser Konfiguration aus keine gültigen Konfigurationen mehr gefunden werden können. Wenn es keine gültigen Konfigurationen gibt, kann mit dieser Bewertung die "beste" ungültige Konfiguration gefunden werden.

## 3.4 Konfigurationsbewertung

Die Resultate der Bewertungsfunktionen werden zu einer Konfigurationsbewertung gemäß 2.1.4 kombiniert. Dazu wurde eine Funktion `fitness( eval_func_type)` eingeführt. Diese wird vom Konfigurierer zur Bewertung einer Konfiguration aufgerufen. Der Parameter `eval_func_type` erlaubt die Auswahl verschiedener Bewertungsfunktionen. Die Funktion `fitness` ermittelt eine Bewertung für eine Konfiguration durch die Kombination der Bewertungen aller Eigenschaften (s. Gleichung 2.4). Die Gewichtungsfaktoren $k_r$ werden vor der Bewertung einer Konfiguration bzw. vor Beginn der automatischen Konfiguration vom Benutzer als Werte in $[0, 100]$ spezifiziert. Eine Gewichtung von 100 bewirkt die volle Einbeziehung einer Anforderung in die Konfigurationsbewertung, während ein Wert von 0 eine Anforderung faktisch ausblendet. Sie hat dann keinen Einfluss auf die Konfigurationsbewertung. Die Gewichtungsfaktoren werden zur Verwendung in `fitness` auf das Intervall $[0, 1]$ skaliert. Der Algorithmus der Funktion `fitness` befindet sich im Anhang A.2. Die Hauptaufgabe dabei ist, für alle spezifizierten nichtfunktionalen Anforderungen die entsprechenden Eigenschaften zu bestimmen, deren Attributmethode aufzurufen und anschließend die Bewertungsfunktion aufzurufen.

## 3.5 Analyse von Eigenschaften

Die Analyse von Eigenschaften einer Konfiguration erfolgt über Attributmethoden. Dabei ist für jede Eigenschaft eine spezielle Methode vorgesehen, welche aus den aktuell in einer Konfiguration verfügbaren Informationen[1] einen Wert, bzw. ein Werteintervall berechnet.

[1] Bindung von Funktionen, Implementierung von Architekturkomponenten etc.

### 3.5.1 WCET einer Konfiguration

Als Indikator für die Verarbeitungsgeschwindigkeit einer Konfiguration soll eine Abschätzung der WCET dienen. Dazu wurde eine Attributmethode "WCET" eingeführt. Die Methode benutzt den in Abschnitt 2.2.4 vorgestellten Algorithmus zur Bestimmung der oberen und unteren Latenzschranke einer Konfiguration. Dazu wird die Methode `SCHEDULE` mit dem Sequenzgraph der Topklasseninstanz der Funktionsschicht aufgerufen. Da die Ablaufplanung auf den WCET der Teilfunktionen beruht, kann die Latenz des ermittelten Ablaufplanes als WCET für das System, bzw. die Konfiguration betrachtet werden. Die Attributmethode liefert das Intervall aus minimaler und maximaler Latenz als Werteintervall für die WCET einer Konfiguration.

Für den Algorithmus zur Ablaufplanung ist die Berechnung der WCET von Basisfunktionen auf bestimmten Ressourcen notwendig. In Abschnitt 2.2.2 wurde beschrieben wie diese Berechnung erfolgen kann. Da die Analyse recht komplex ist, besonders aufgrund der Notwendigkeit eines eigenen Berechnungsmodells für jede einzelne Ressource, wurden die Informationen über die WCET von Basisfunktionen manuell in der Wissensbasis zur Verfügung gestellt. Dazu wurden die Regeln in Tabelle 3.2 für die Wissensbasis aufgestellt. Abbildung 3.1 zeigt ein Beispiel für eine Wissensbasis, welche die Regeln für die Attributmethode WCET befolgt.

### 3.5.2 Kosten einer Konfiguration

Für additive Eigenschaften wurde die Attributmethode "SUM" eingeführt. Wird ein Attribut $a$ mit Attributmethode SUM definiert, so iteriert SUM über alle Parts der Klasseninstanz und summiert alle Attribute mit dem gleichen Namen wie $a$.

Um die Kosten einer Konfiguration zu modellieren, wurde des Weiteren die Attributmethode "COST" eingeführt. Als Kosten einer Konfiguration gelten dabei, analog zu den Definitionen in Abschnitt 2.3, die Basiskosten der Ressourcen der Architektur. Daher bestimmt "COST" zunächst die Topklasse der Architekturschicht. Ist diese noch nicht vorhanden, weil für die Konfiguration noch keine Architektur ausgewählt wurde, so meldet "COST" dem

| Regel | Beschreibung |
|---|---|
| Definition der Eigenschaft für die WCET | Die Eigenschaft, welche die WCET einer Konfiguration repräsentiert, ist in der Topklasse der Funktionsschicht mit der Attributmethode "WCET" zu definieren. Das Werteintervall enthält als Minimum die minimal mögliche Latenz aller gültigen Konfigurationen und als Maximum die maximale Latenz aller gültigen Konfigurationen. |
| WCET bei Nicht-Basisfunktionen | Jede Klasse der Funktionsschicht, welche keine Basisfunktion darstellt, muss ein Attribut mit dem Namen "WCET" besitzen, dessen Werteintervall die minimal mögliche Latenz der Funktion und die maximal mögliche Latenz dieser Funktion in allen gültigen Konfigurationen definiert. |
| WCET bei Basisfunktionen | Jede Klasse der Funktionsschicht, welche eine Basisfunktion darstellt, muss für jede mögliche Bindung ein Attribut mit dem Namen "WCET_" + <Name der Ressource> besitzen, welches ein Werteintervall für die WCET dieser Basisfunktion auf der entsprechenden Ressource beschreibt. |

Tabelle 3.2: Regeln für die Wissensbasis für die Attributmethode WCET.

Berechnungsmechanismus von RODOS, dass kein Wertintervall berechnet werden konnte. RODOS sorgt in diesem Fall dafür, dass das in der Wissensbasis definierte Wertintervall übernommen wird. Ist die Topklasse der Architekturschicht vorhanden, so ermittelt "COST" die Summe der Parts dieser Klasseninstanz, also die Summe der Basiskosten. Die Voraussetzung dazu ist, dass die Werte der Basiskosten jeder einzelnen Ressource korrekt bestimmt werden. Die Basiskosten werden durch ein additives Attribut mit dem gleichen Namen wie das Attribut für die Kosten der Konfiguration in der Topklasse der Funktionsschicht repräsentiert. Dieses Attribut wird mit der Attributmethode "SUM" definiert und muss in allen Komponenten der Implementationsschicht vorhanden sein, um die korrekte Berechnung der Basiskosten zu gewährleisten. Eine Übersicht der Regeln für die Wissensbasis

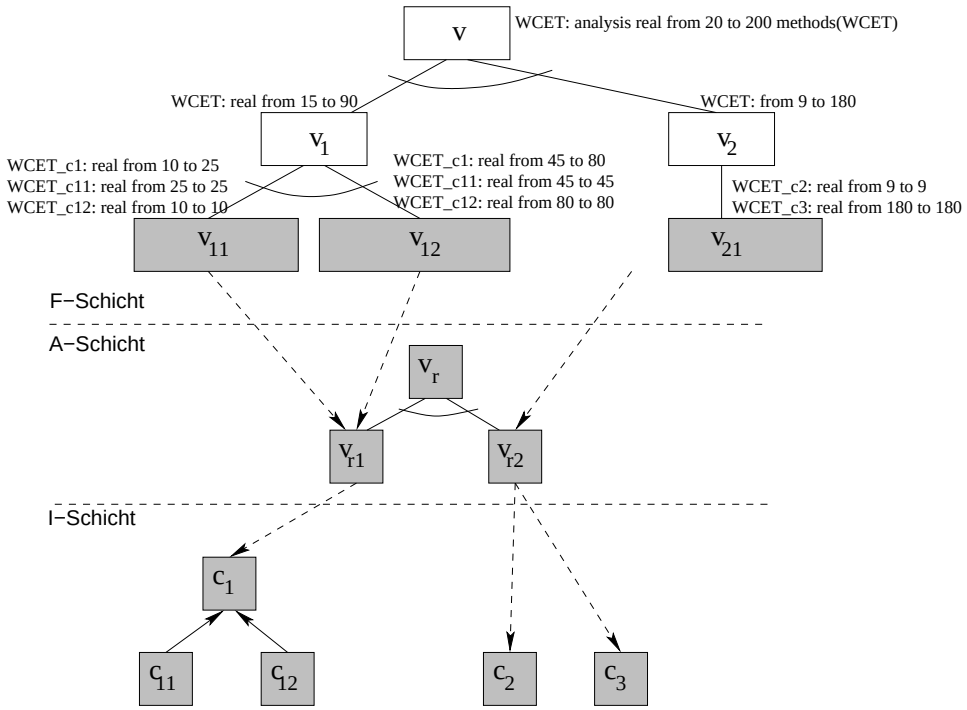


Abbildung 3.1: Beispiel für WCET Attribute in einer Wissensbasis.

zur Berechnung der Kosten gibt Tabelle 3.3. Abbildung 3.2 zeigt ein Beispiel für eine Wissensbasis, welche die Regeln für die Attributmethode COST befolgt.

### 3.5.3 Energieverbrauch einer Konfiguration

Der Energieverbrauch, sowie alle weiteren additiven Eigenschaften einer Konfiguration, können analog zu den Kosten modelliert werden. Es gelten die gleichen Regeln wie für die Berechnung der Kosten.

### 3.5.4 Visualisierung der Analyseergebnisse

Um dem Entwerfer einen Überblick über die Eigenschaften von Konfigurationen zu bieten, kann der Entwurfsraum grafisch dargestellt werden. Dazu wurde unter dem Menü “Tools” ein neuer Menüpunkt “Design Space Chart” eingefügt. Über diesen gelangt der Entwerfer zu einem Dialog, in welchem ein Diagramm dargestellt wird. Die Achsen des Diagramms stehen für aus-

| Regel | Beschreibung |
|---|---|
| Definition der Eigenschaft für die Kosten | Die Eigenschaft, welche die Kosten einer Konfiguration repräsentiert, ist in der Topklasse der Funktionsschicht mit der Attributmethode "COST" zu definieren. |
| Definition der Basiskosten | In allen Klassen der Implementationsschicht, sowie in allen Klassen unterhalb der Topklasse der Architekturschicht müssen Attribute mit dem gleichen Namen wie das Eigenschaftsattribut vorhanden sein. Diese müssen mit der Attributmethode "SUM" definiert sein. |
| Zusatzkosten | Zusatzkosten werden durch Attribute in den Basisfunktionen definiert. Für jede Bindungsmöglichkeit einer Basisfunktion muss ein Attribut, dessen Name sich aus dem Namen des Eigenschaftsattributes und dem Namen der Ressource zusammensetzt, vorhanden sein. |

Tabelle 3.3: Regeln für die Wissensbasis für die Attributmethode COST.

wählbare Eigenschaften. Jede in der aktuellen Sitzung des Entwurfswerkzeuges vorhandene Konfiguration wird im Diagramm als Rechteck dargestellt. Die Dimensionen des Rechtecks spiegeln die Intervalle der Eigenschaften der Konfiguration wieder. So kann beispielsweise für alle Konfigurationen der Entwurfsraum bezüglich der Eigenschaften "Kosten" und "Energieverbrauch" visualisiert werden. Befindet sich der Mauszeiger über einem Rechteck, werden dem Entwerfer weitere Informationen über die entsprechende Konfiguration angezeigt. Abbildung 3.3 zeigt den Dialog mit einem Beispiel für einen Entwurfsraum. In diesem Beispiel sind insgesamt sieben Konfigurationen zu erkennen. Die sechs kleineren Rechtecke stellen vollständige Konfigurationen dar. Das größere Rechteck stellt eine unvollständige Konfiguration dar. Sie ist eine Vorgängerkonfiguration der anderen Konfigurationen, weshalb sie diese einschließt.

Zur Darstellung von Ablaufplänen eignen sich Gantt-Diagrame. Unter dem Menü "Tools" befindet sich ein neuer Menüpunkt "View Schedule", über

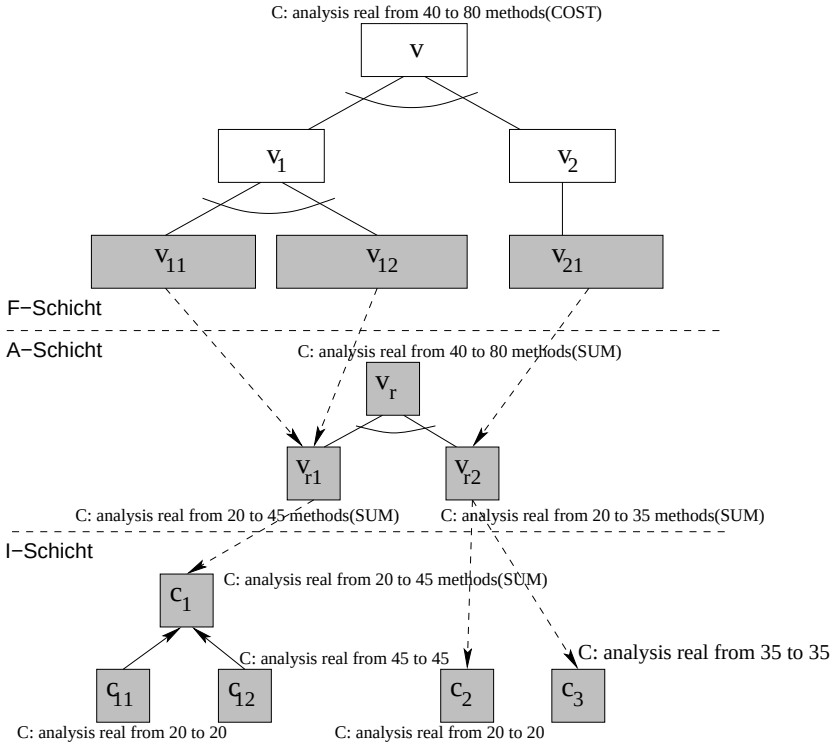


Abbildung 3.2: Beispiel für COST Attribute in einer Wissensbasis.

welchen ein Dialog geöffnet wird. Dieser enthält die Darstellung des Ablaufplanes der aktuellen Konfiguration als Gantt-Diagramm. Jedes Rechteck steht für eine Basisfunktion, deren Start- und Endzeitpunkte durch die Ausmaße des Rechtecks in x-Richtung ablesbar sind. Der y-Achse sind die Ressourcen der Konfiguration zugeordnet. Die Position des Rechtecks in y-Richtung gibt also die Bindung der entsprechenden Basisfunktion wieder. Bewegt der Entwerfer den Mauszeiger über ein Rechteck, so werden Informationen wie Name, Start- und Endzeit der Basisfunktion angezeigt. Abbildung 3.4 zeigt den Dialog.

Um die funktionalen Zusammenhänge der Konfiguration zu betrachten, kann der Sequenzgraph graphisch dargestellt werden. Dazu befindet sich im Dialog “Explore Configuration” ein Knopf “CDFG”. Über diesen kann der Sequenzgraph in eine Datei mit speziellem Format exportiert werden. Diese wird von dem Tool *XVCG* gelesen und graphisch angezeigt. Abbildung 3.5 zeigt das Beispiel eines Sequenzgraphen. Der Sequenzgraph zeigt, wie die

Gesamtfunktion in zwei voneinander unabhängige Funktionen unterteilt ist. Jede der Teilfunktionen besteht wiederum aus mehreren Funktionen. Die blau gefärbten Rechtecke stellen Basisfunktionen, also nicht mehr unterteilte Teilfunktionen dar. Die Teilfunktion "TH" besteht aus einem Verzweigungsknoten, dessen Teilzweige je eine Basisfunktion beinhalten.

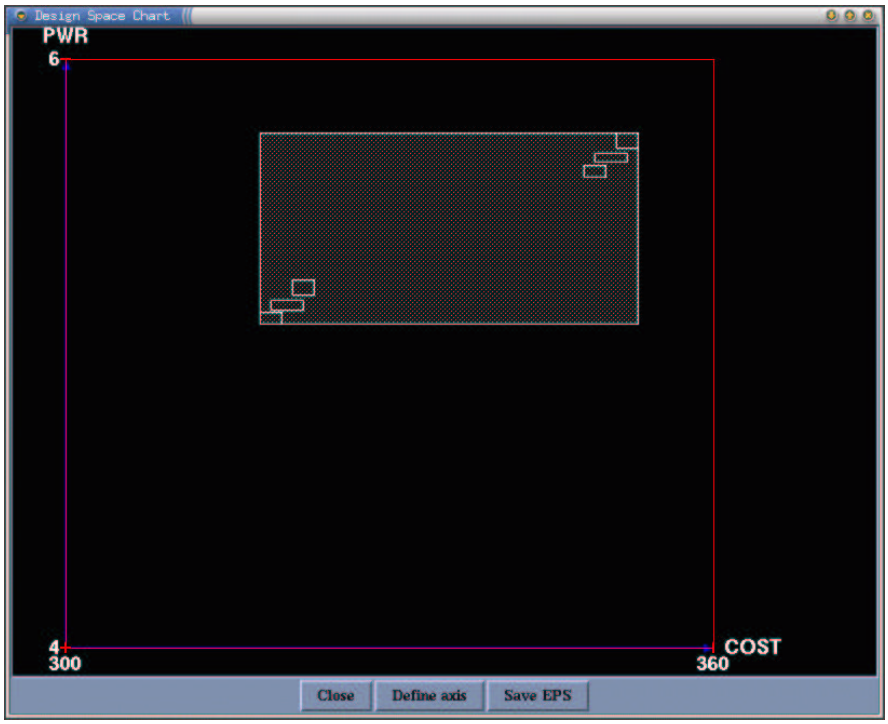


Abbildung 3.3: Der Dialog "Design Space Chart" zur Visualisierung des Entwurfsraumes.

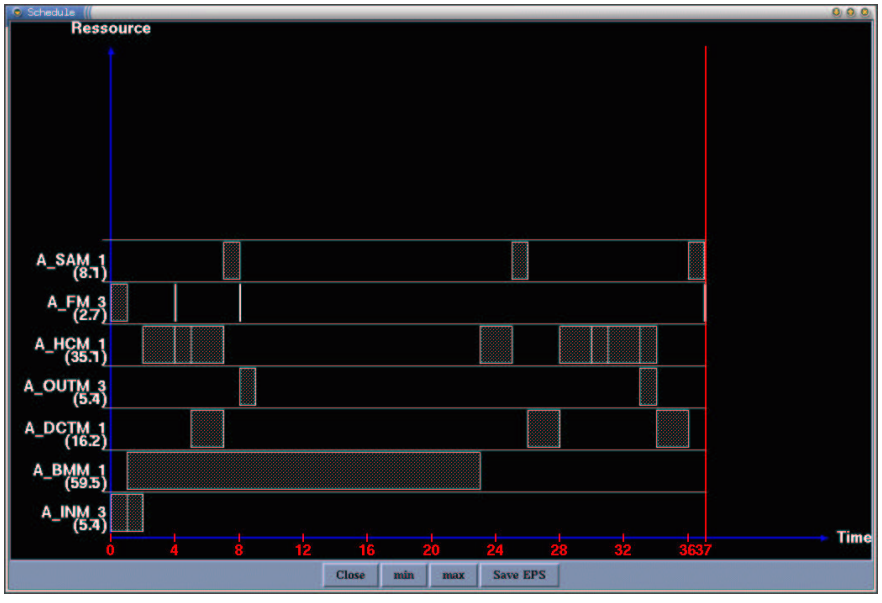


Abbildung 3.4: Der Dialog "Schedule" zur Visualisierung von Ablaufplänen.

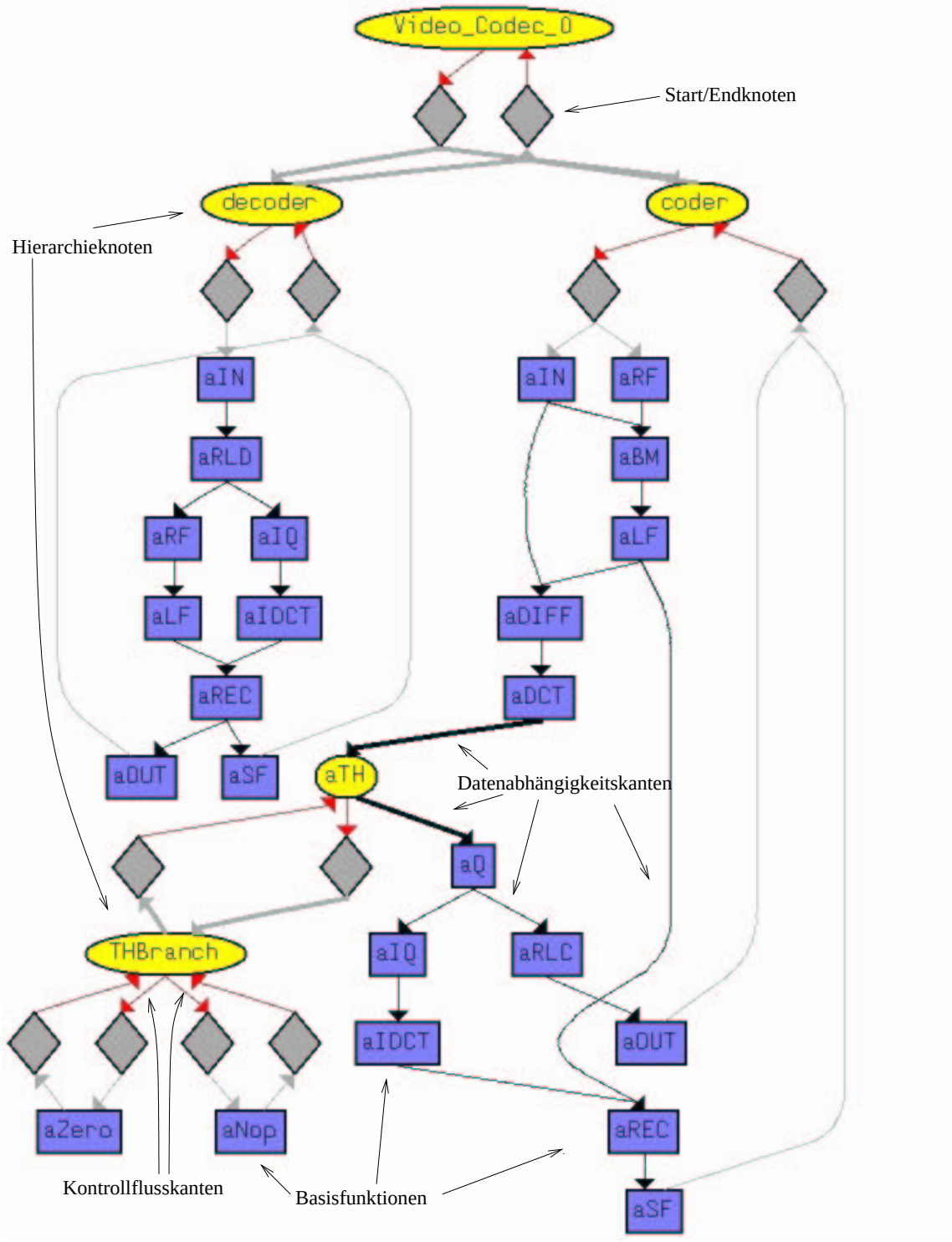


Abbildung 3.5: Beispiel eines Sequenzgraphen.

# Kapitel 4

# Implementierung der Verfahren

## 4.1 Repräsentation des Kontroll- und Datenflusses

Die Notation efolgt nach einer Grammatik (siehe Anhang A.1) die es erlaubt, Datenabhängigkeiten sowie Kontollfluss gemeinsam zu Beschreiben.

Alle Klassen und Prozeduren für das Einlesen und Exportieren von Sequenzgraphen befinden sich in den Dateien `cdfg.h` und `cdfg.cc` im Verzeichnis `Database`.

Die Beschreibung des Sequenzgraphen wird beim Laden der Wissensbasis mit den anderen Bestandteilen einer Wissensklasse eingelesen. Dazu wurde in der Klasse `Concept` eine Methode `parse_cdfg()` implementiert, welche das Einlesen und Überprüfen der Sequenzgraphbeschreibung übernimmt. Dabei werden die verwendeten Bezeichner auf Gültigkeit überprüft. Die interne Repräsentation des Sequenzgraphen ist ein LEDA-Graph. Dieser existiert nach dem Einlesen als Mitglied der Klasse `Concept`.

Die Methode `parse_cdfg(...)` erzeugt einen neues Objekt vom Typ `CDFG_PARSER` und ruft dessen Methode `parse()`. Dies führt zum Aufbau eines Parsebaumes, dessen Knoten abgeleitete Klassen von `CDFG_PARSE_NODE` sind. Jede dieser Klassen implementiert die abstrakte Methode `parse()` um sich selbst zu parsen. Abbildung 4.1 zeigt eine Übersicht über die verwendete Klassenhierarchie.

Nach dem Parsen der Sequenzgraphbeschreibung erfolgt die Erzeugung

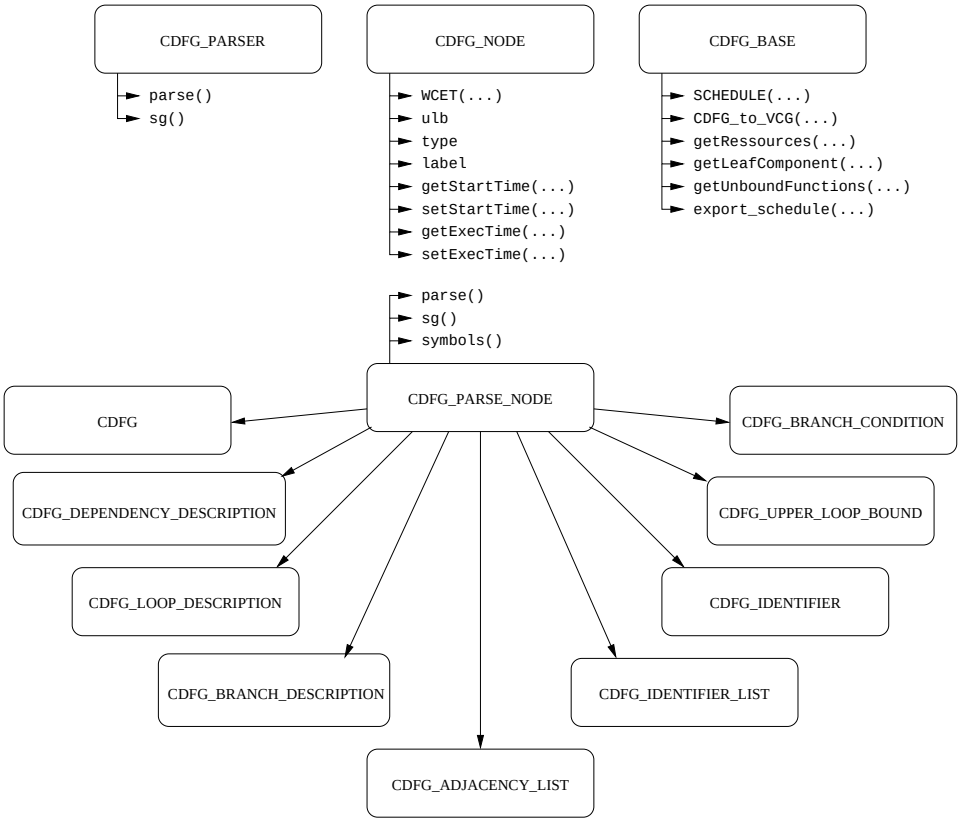


Abbildung 4.1: Klassenhierarchie `cdfg.h`.

des Sequenzgraphen durch einen Aufruf der Methode `sg(...)` des Parsers. Diese liefert einen Zeiger auf einen LEDA-Graphen, der in der Klasse `Concept` gespeichert wird.

Sequenzgraphen können visualisiert werden. Dazu wird das Tool *XVCG* zum Anzeigen einer Graphbeschreibung verwendet. Die Sequenzgraphen können durch den Aufruf der Methode `CDFG_to_VCG(...)` in eine solche Graphbeschreibung exportiert werden.

## 4.2 Konfigurationsbewertung

Die Konfigurationsbewertung erfolgt durch Aufruf der Methode `fitness(...)` der Klasse `Costruction`. Der Eingangsparameter vom Typ `eval_func_type` spezifiziert die zu verwendende Bewertungsfunktion. Der Rückgabewert ist vom Typ `double` und auf das Intervall $[0, 1]$ beschränkt. Der Algorithmus in Abschnitt A.2 beschreibt das Verhalten von `fitness`.

## 4.3 Bewertungsfunktionen

In der Klasse `Costruction` befinden sich Methoden für die in Abschnitt 3.3 beschriebenen Bewertungsfunktionen. Die Methoden haben je zwei Eingangsparameter vom Typ `Attr_Inst*`. Der Parameter `*req_attr` repräsentiert das Anforderungsattribut und der Parameter `*attr` repräsentiert das Eigenschaftsattribut. Der Rückgabewert vom Typ `double` liegt im Intervall $[0, 1]$. Im Anhang, Abschnitt A.3, befinden sich die Algorithmen zu den Bewertungsfunktionen.

## 4.4 Attributmethoden

Die Attributmethode "WCET" wird durch die Methode `eval_wcet` implementiert. Die Methode `eval_wcet` initialisiert die Datenstrukturen für die Ablaufplanung und ermittelt je einen Ablaufplan mit unterer und oberer Latenzschranke. Anschließend wird das Werteintervall des mit der Methode verbundenen Attributes gesetzt. Die ist untere Intervallgrenze gleich der

unteren Latenzschranke und die obere Intervallgrenze gleich der oberen Latenzschranke.

Die Attributmethode "COST" wird durch die Methode `eval_cost` implementiert. Beide Methoden sind Mitglieder der Klasse `Analysis_Engine`. Die Algorithmen der Attributmethoden befinden sich im Anhang, Abschnitt A.5.

## 4.5 Ablaufplanung

Die Methode `SCHEDULE` implementiert den Algorithmus 1. Über den Eingangsparameter `minmax` wird das Verhalten von `SCHEDULE` gesteuert. Ist der Parameter `minmax` gleich "true", so erzeugt die Methode einen Ablaufplan mit unterer Latenzschranke. Anderenfalls erzeugt die Methode einen Ablaufplan mit oberer Latenzschranke. Der Rückgabewert vom Typ `Inter_Val` enthält das Zeitintervall in dem alle Funktionen ausgeführt werden. Die Latenz eines Ablaufplanes ist damit gleich der Länge dieses Intervalls.

## 4.6 Visualisierung

Die Dialoge "Schedule" und "Design Space Chart" sind in TCL/Tk implementiert. Die Kommunikation mit dem Entwurfswerkzeug erfolgt über den conshell-Befehl `autoconfig`.

| Methode | Klasse | Datei |
|---|---|---|
| `fitness` | `Construction` | cf_construction.cc |
| `interval_match` | `Construction` | cf_construction.cc |
| `interval_overlap` | `Construction` | cf_construction.cc |
| `interval_distance` | `Construction` | cf_construction.cc |
| `eval_wcet` | `Analysis_Engine` | analysis_engine.cc |
| `eval_cost` | `Analysis_Engine` | analysis_engine.cc |
| `SCHEDULE` | `CDFG_BASE` | cdfg.cc |
| `export_schedule` | `CDFG_BASE` | cdfg.cc |
| `CDFG_to_VCG` | `CDFG_BASE` | cdfg.cc |
| `design_space_chart` | - | tk_chart.tcl |
| `view_schedule` | - | tk_schedule.tcl |

Tabelle 4.1: Übersicht der wichtigsten implementierten Methoden.

# Kapitel 5

# Anwendung und Auswertung der Verfahren zur Analyse und Bewertung

## 5.1 Fallbeispiel Videocodec

Das Fallbeispiel soll zeigen, wie die entwickelten Verfahren zur Analyse und Bewertung von Konfigurationen angewendet werden. Dazu wurde die Wissensbasis videocodec.clint entwickelt. Sie lehnt sich an ein komplexes Beispiel aus [Teich 1997, S.414-426]. Das dort beschriebene System ist ein Video-Codec nach der H.261-Norm. Abbildung 5.1 zeigt ein Blockdiagramm des Verhaltens eines solchen Video-Codec.

### 5.1.1 Wissensbasis

Die Wissensbasis wurde mit dem Ziel entwickelt, die Verfahren zur Analyse und Bewertung zu demonstrieren. Auf eine exakte Funktionsbeschreibung der Basisfunktionen wurde dabei verzichtet. In die Architekturschicht wurden drei unterschiedliche Architekturen aufgenommen. Für zwei davon gibt es jeweils nur eine Bindungsmöglichkeit, d.h. jede Basisfunktion kann an genau eine Ressource dieser Architekturen gebunden werden. Bei der dritten Architektur könnnen sechs Basisfunktionen auf je 2 unterschiedliche Ressourcen gebunden werden. Das ergibt eine Zahl von $2^6 = 64$ unterschiedlichen Bindungsmöglichkeiten.

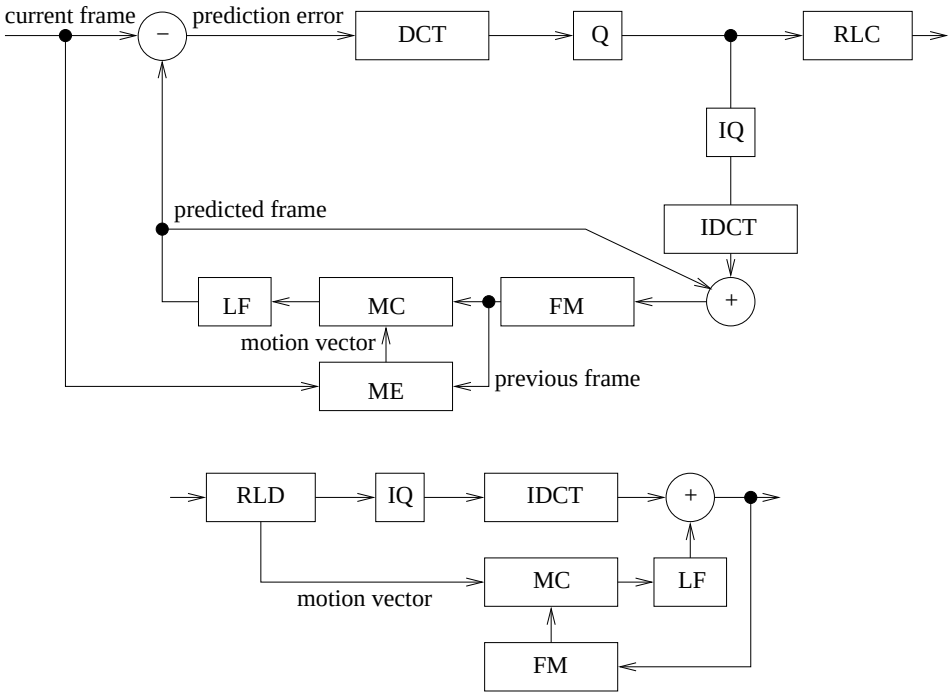


Abbildung 5.1: Verhaltensbeschreibung eines Video_Codec nach der H.261-Norm (oben: Encoder, unten: Decoder).

In die Wissensbasis wurden drei Eigenschaften aufgenommen. Diese sind die Ausführungszeit "WCET", die Kosten "COST" und der Energieverbrauch "PWR". Die Eigenschaft "WCET" wurde mit der Attributmehode "WCET" definiert. Die Eigenschaften "COST" und "PWR" sind additive Eigenschaften und daher mit der Attributmethode "COST" definiert. Auf die Modellierung von Zusatzkosten wurde verzichtet. Eine genauere Beschreibung der Wissensbasis befindet sich in Anhang B.

### 5.1.2 Demonstration anhand manueller Konfiguration

Die folgenden Abbildungen zeigen die Arbeitsoberfläche von RODOS. Ausgehend von der Anfangskonfiguration in Abbildung 5.2 werden sukzessive Konfigurationsschritte angewandt, welche die Konfiguration allmählich vervollständigen. Unter den Abbildungen der Arbeitsoberfläche ist jeweils ein Ablaufplan für die entsprechende Konfiguration zu sehen. Aus Platzgründen ist nur der Ablaufplan für die obere Latenzschranke zu sehen. Der Ablauf-

plan für die untere Latenzschranke ist diesem meist sehr ähnlich, jedoch mit geringerer Latenz.

In der Ausgangskonfiguration befindet sich lediglich die Topklasseninstanz "Video_Codec_0". Der Ablaufplanungsalgotithmus ermittelt den Ablaufplan in Abbildung 5.3, welcher nur eine ungebundene Funktion zeigt. Mehr Informationen sind noch nicht in der Konfiguration enthalten. Tabelle 5.1 enthält die Werteintervalle der Anforderungen und Eigenschaften der Ausgangskonfiguration. Die mit "Fitness" angegebene Konfigurationsbewertung wurde mit der Berwertungsfunktion `interval_overlap` ermittelt.

| Eigenschaft | $\min_{Anf.}$ | min | max | $\max_{Anf.}$ |
|---|---|---|---|---|
| WCET | 0 | 36 | 305 | 80 |
| COST | 100 | 153 | 385 | 360 |
| PWR | 0 | 5.1 | 30.85 | 40 |
| Fitness | 68% | | | |

Tabelle 5.1: Eigenschaften der Ausgangskonfiguration.

Durch Dekomposition der Funktion "Video_Codec_0" in ihre Teilfunktionen "H261_Coder_0" und "H261_Decoder_0" ergibt sich eine neue Konfiguration, deren Eigenschaften unverändert die Werte der Ausgangskonfiguration haben. Der Ablaufplan dieser neuen Knofiguration ist aber mit der Dekomposition verfeinert worden und enthält 2 Funktionen. Durch weitere Dekomposition der beiden Teilfunktionen vervollständigt sich die Funktionsschicht. Die Eigenschaften haben immer noch die Werte der Ausgangskonfiguration. Abbildung 5.4 zeigt die vervollständigte Funktionsschicht. Der Ablaufplan enthält nun alle Basisfunktionen der Funktionsschicht. Es sind aber noch keine Bindungen an Ressourcen vorhanden, so das der Ablaufplan alle Funktionen hintereinander plant, wie in Abbildung 5.5 zu sehen ist. Der Ablaufplan für die untere Latenzschranke würde an dieser Stelle alle Funktionen parallel geplant zeigen, woraus sich die minimale WCET von 36, wie in Tabelle 5.1 ergibt.

Da nun alle Basisfunktionen vorhanden sind, kann die Auswahl einer Architektur erfolgen. Danach erfolgt die Dekomposition der Topklasseninstanz der Architektur, so dass für alle Ressourcen Klasseninstanzen erzeugt wer-

den. Anschließend kan die Bindung von Funktionen an Ressourcen erfolgen. Abbildung 5.6 zeigt die Konfiguration nach der Ausführung dieser Schritte. Als Architektur wurde "ARCH_TOP_1" (s. Abbildung B.3) gewählt. Alle Ressourcen werden durch Klasseninstanzen der Implementationsschicht implementiert. Alle Bindungen wurden durchgeführt. Die Konfiguration ist fast vollständig, lediglich die Ressource "H_BM_0" kann noch spezialisiert werden, was aber nicht weiter betrachtet werden soll. Der Ablaufplan in Abbildung 5.7 zeigt nun die zeitliche Verteilung der Funktionen auf den unterschiedlichen Ressourcen. Tabelle 5.2 zeigt die Werteintervalle der Anforderungen und Eigenschaften der Konfiguration.

| Eigenschaft | $\min_{Anf.}$ | min | max | $\max_{Anf.}$ |
|---|---|---|---|---|
| WCET | 0 | 37 | 49 | 80 |
| COST | 100 | 318 | 353 | 360 |
| PWR | 0 | 5.1 | 5.75 | 40 |
| Fitness | 100% | | | |

Tabelle 5.2: Eigenschaften von "ARCH_TOP_1".

In Abbildung 5.8 ist die Konfiguration nach Auswahl der Architektur "ARCH_TOP_2" (s. Abbildung B.3). Bei dieser Architektur ist eine Vielzahl unterschiedlicher Bindungen möglich, von denen eine angewendet wurde. In dieser Konfiguration können die Ressourcen "H_PROCESSOR_0" und "H_DSP" noch spezialisiert werden. Bei der verwendeten Bindung ermittelt der Ablaufplanungsalgorithmus den Ablaufplan in Abbildung 5.9. Im Vergleich zum Ablaufplan in Abbildung 5.7 kann man beobachten, das duch die geringere Anzahl Ressourcen weniger Nebenläufigkeit herrscht. Zusätzlich sind die Ausführungszeiten der Funktionen länger, was insgesamt zu einer größeren Latenz führt.

| Eigenschaft | $\min_{Anf.}$ | min | max | $\max_{Anf.}$ |
|---|---|---|---|---|
| WCET | 0 | 104 | 116 | 80 |
| COST | 100 | 353 | 385 | 360 |
| PWR | 0 | 15.7 | 30.85 | 40 |
| Fitness | 0% | | | |

Tabelle 5.3: Eigenschaften von "ARCH_TOP_2".

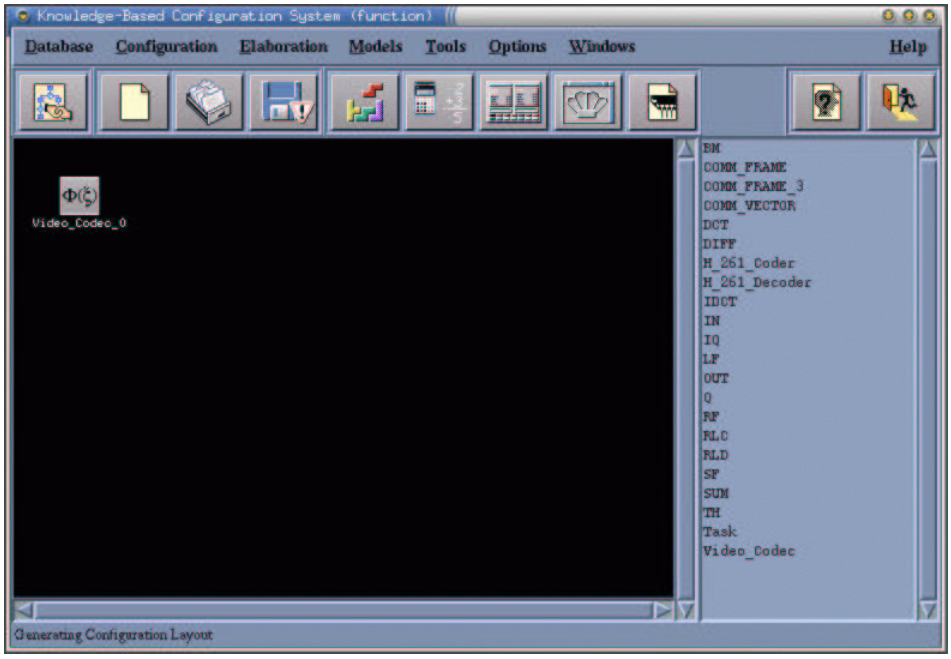


Abbildung 5.2: Ausgangskonfiguration: Video_Codec_0.

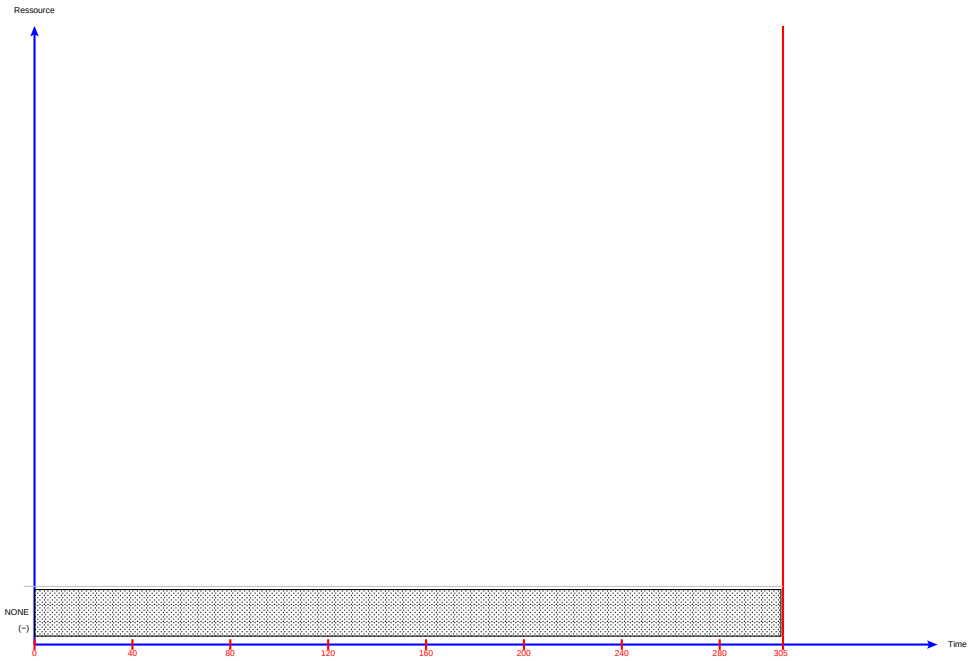


Abbildung 5.3: Ablaufplan der Ausgangskonfiguration.

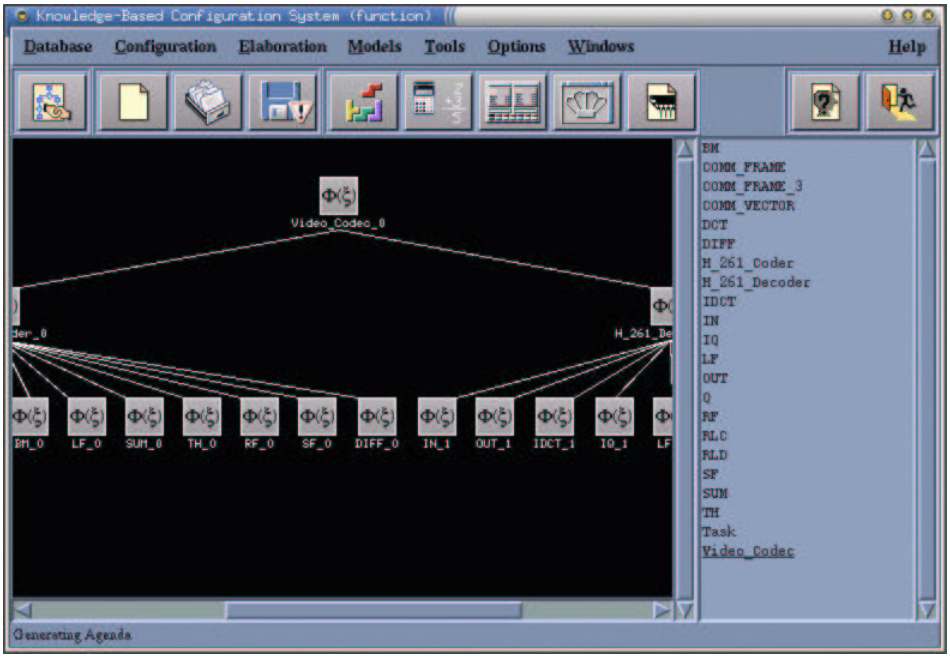


Abbildung 5.4: Konfiguration mit der vervollständigten Funktionsschicht.

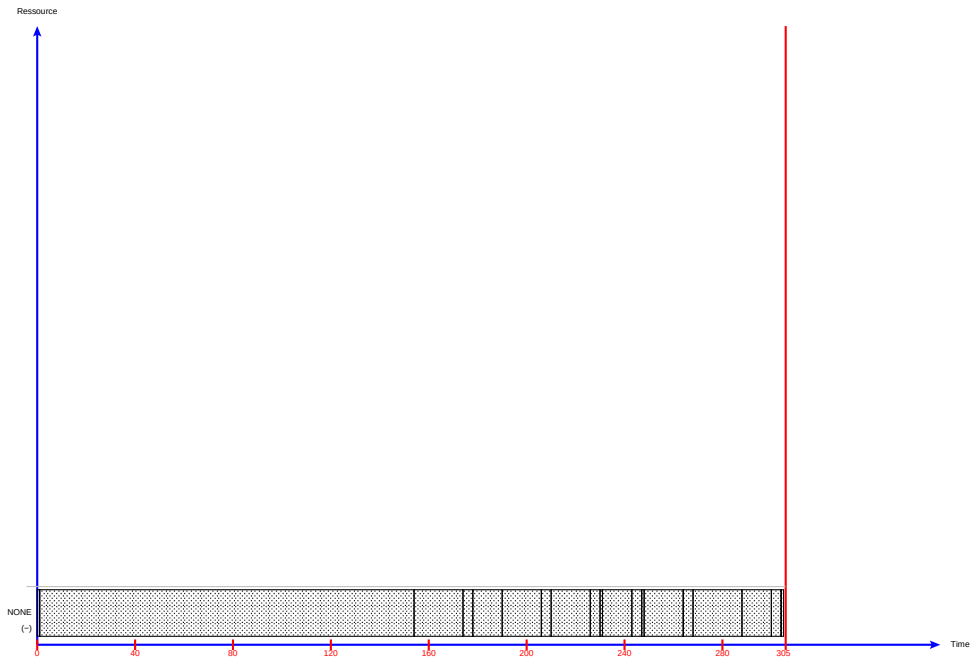


Abbildung 5.5: Ablaufplan, es ist noch keine der Basisfunktionen an eine Ressource gebunden.

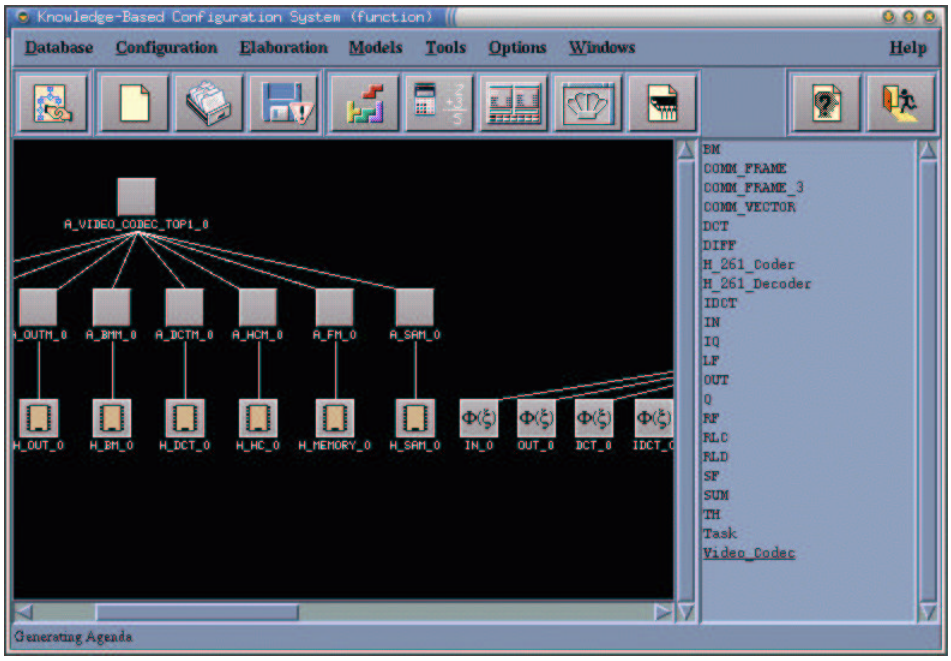


Abbildung 5.6: Eine Konfiguration mit der Architektur “ARCH_TOP_1”.

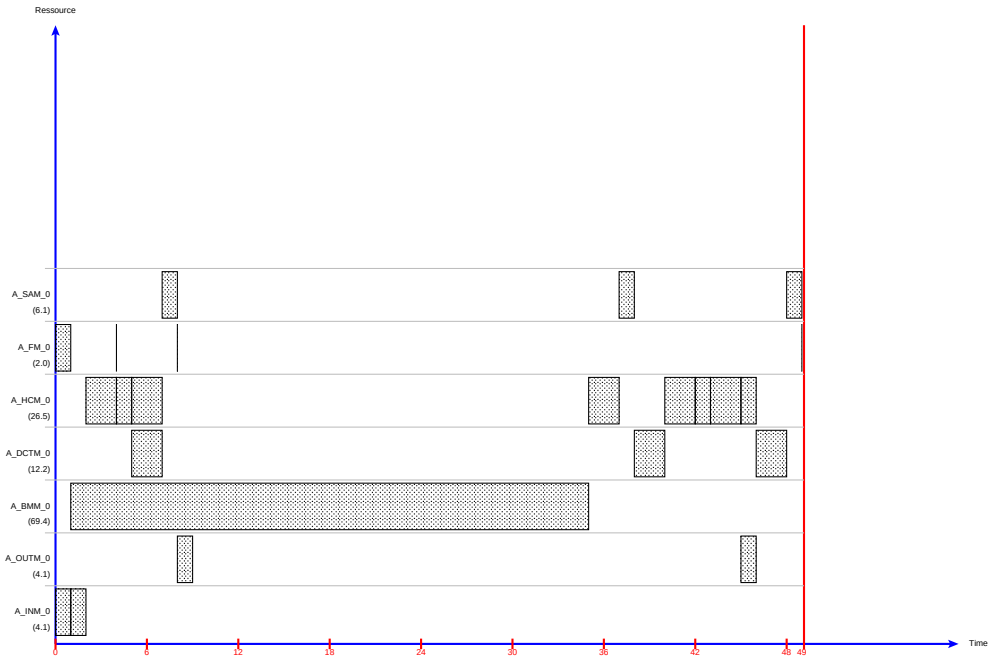


Abbildung 5.7: Ablaufplan der Basisfunktionen auf “ARCH_TOP_1”.

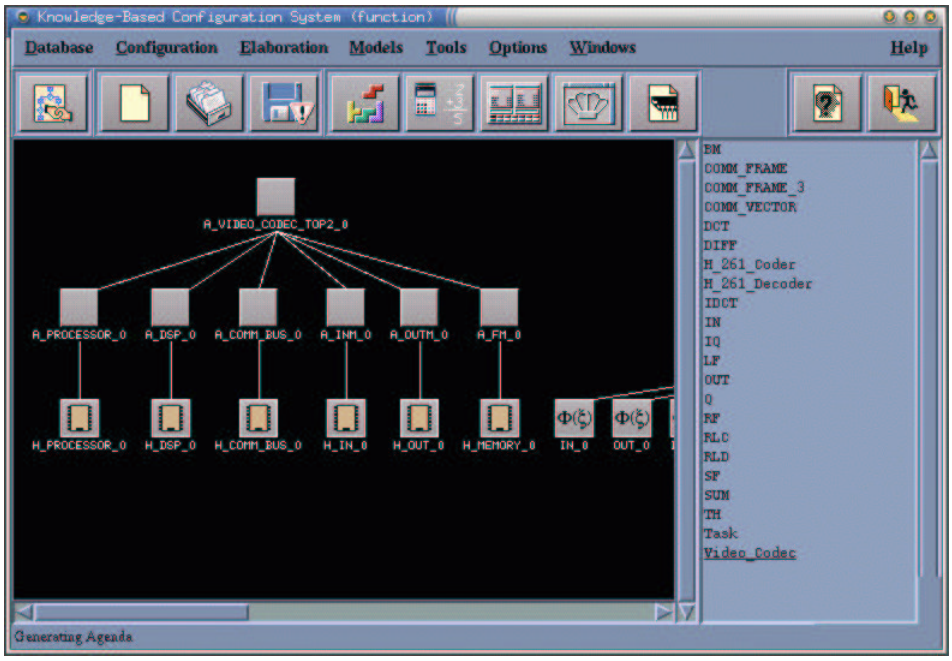


Abbildung 5.8: Eine Konfiguration mit der Architektur “ARCH_TOP_2”.

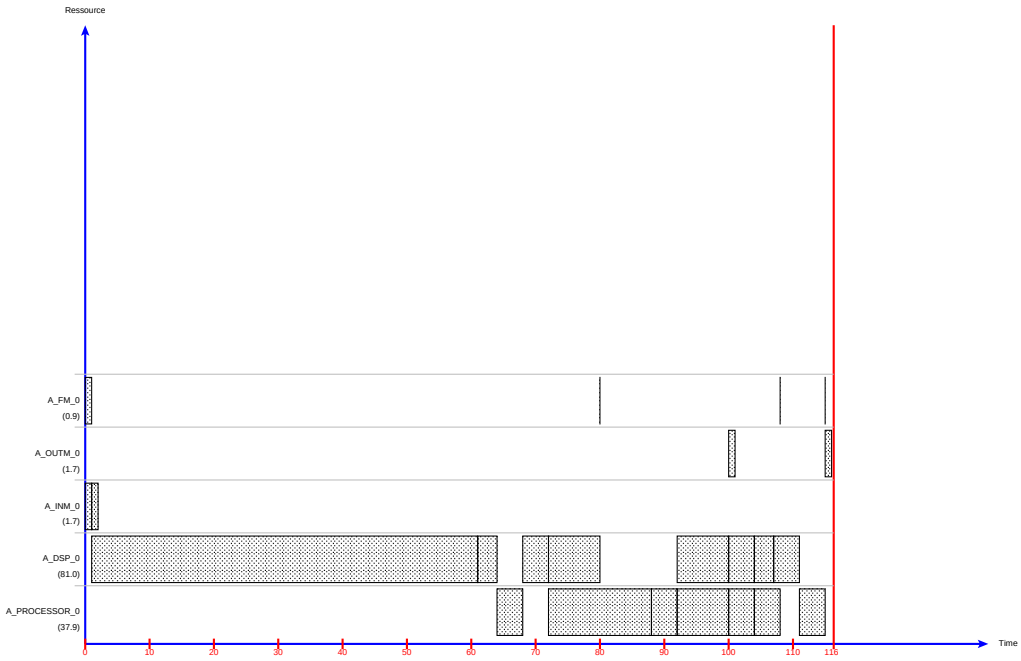


Abbildung 5.9: Ablaufplan der Basisfunktionen auf “ARCH_TOP_2”.

# Kapitel 6

# Zusammenfassung

Um die Komplexität des Entwurfs eingebetteter Systeme zu beherrschen, ist der Entwerfer auf computergestützte Werkzeuge angewiesen. Da selbst diese bei der Lösung vieler Probleme an ihre Grenzen stoßen, ist es von großem Interesse die Werkzeuge mit effizienten Mechanismen auszustatten. Ein Ansatz dazu ist der wissensbasierte Entwurf eingebetteter Systeme, welchen das Entwurfswerkzeug RODOS verfolgt.

In RODOS existiert ein automatisierter Konfigurationsmechanismus. Dieser benötigt bei der Suche nach Zielkonfigurationen die Möglichkeit, Konfigurationen zu bewerten, um sie miteinander zu vergleichen und festzustellen, ob sie den spezifizierten nichtfunktionalen Anforderungen genügen. In der Diplomarbeit wurden Methoden und Verfahren entwickelt, welche die Bewertung und Analyse der nichtfunktionalen Eigenschaften und Anforderungen von Konfigurationen ermöglichen.

Die Definition von Eigenschaften und Anforderungen erfolgt mit Hilfe von Attributen. Speziell eingeführte Attributmethoden ermöglichen die Analyse der Eigenschaften. Bewertungsfunktionen bewerten dann die Anforderungen bezüglich der Eigenschaften. Die Konfigurationsbewertung kombiniert die Bewertungen der Anforderungen zu einer Gesamtbewertung einer Konfiguration.

Als Teil der Analyse wurde ein Ablaufplanungsalgorithmus entwickelt. Dieser ermöglicht Aussagen über das zeitliche Verhalten von Konfigurationen, auch wenn sie noch unvollständig sind. Die Analyseergebnisse können

grafisch dargestellt werden. Der Dialog “Design Space Chart” eignet sich zur Visualisierung der Bewertung einer Konfiguration mit der Übersicht über die Werte der Eigenschaften. Das zeitliche Verhalten wird im Dialog “Schedule” veranschaulicht, welcher den Ablaufplan einer Konfiguration anzeigt. Die Anwendbarkeit der Verfahren wurde am Fallbeispiel “Videocodec” gezeigt.

# Literaturverzeichnis


**Buchenrieder 2001** BUCHENRIEDER, Klaus: Klaus Buschenrieder(Hrsg.), ItPress, 2001. – ISBN 3-929814-08-0

**Coffman 1976** COFFMAN, E.G.: *Computer and Job-Scheduling Theory.* John Wiley and Sons, New York, 1976

**Ecker 2001** ECKER, Wolfgang: Hardware-basierter Hardware-Entwurf. In: *4. GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen* Bd. 1. Dieter Monjau(Hrsg.), MoPress, 2001. – ISBN 3-00-007439-2

**Engblom und Ermedahl 2000** ENGBLOM, J. ; ERMEDAHL, A.: Modeling complex flows for worst-case execution time analysis. In: *Proc. of 21st IEEE Real-Time Systems Symposium (RTSS'00)*, URL `http://citeseer.nj.nec.com/engblom00modeling.html`, November 2000. – Accepted for publication.

**Engblom u. a. 2000** ENGBLOM, Jakob ; ERMEDAHL, Andreas ; SJODIN, Mikael ; GUSTAFSSON, Jan ; HANSSON, Hans: *Worst-Case Execution-Time Analysis for Embedded Real-Time Systems.* 2000. – URL `http://citeseer.nj.nec.com/engblom00worstcase.html`

**Förster 2001** FÖRSTER, Stefan: *Optimierung des Konfigurierers im Entwurfssystem RODOS*, TU Chemnitz, Diplomarbeit, Februar 2001

**Jamshidi 1996** JAMSHIDI, Mohammad: Large-Scale Systems: Modelling, Control, and Fuzzy Logic. In: *Prentice Hall series on environmental and*

*intelligent manufacturing systems* Bd. 8. Prentice Hall, 1996. – ISBN 0-13–125683-1

**Li und Malik 1995** Li, Y-T. S. ; Malik, S.: Performance Analysis of Embedded Software Usin Implicit Path Enumeration. In: *ACM Workshop on Languages, Compilers, and Tools for Real-Time Systems*, May 1995

**Lindgren 2000** Lindgren, Markus: *Using Measurements to Derive the Worst-Case Execution Time.* 2000. – URL `http://citeseer.nj.nec.com/412385.html`

**Malik u. a. 1997** Malik, Sharad ; Martonosi, Margaret ; Li, Yau-Tsun S.: Static Timing Analysis of Embedded Software. In: *Design Automation Conference*, URL `http://citeseer.nj.nec.com/malik97static.html`, 1997, S. 147–152

**Micheli 1994** Micheli, G. D.: *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, New York, 1994

**Monjau und Sporer 2000** Monjau, Dieter ; Sporer, Matthias: *RODOS.* : TU Chemnitz (Veranst.), 2000. – URL `http://herkules.informatik.tu-chemnitz.de/wisyra/rodos.html`. – Online-Dokumentation

**Mueller und Wegener 1998** Mueller, F. ; Wegener, J.: A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. In: *IEEE Real-Time Technology and Applications Symposium*, URL `http://citeseer.nj.nec.com/mueller98comparison.html`, June 1998, S. 179–188

**Park 1992** Park, C.Y.: *Predicting Deterministic Execution Times of Real-Time Programs*, University of Washington, Seattle, Dissertation, August 1992

**Paulin und Knight 1989** PAULIN, P.G. ; KNIGHT, J.P.: Force-directed scheduling for the behavioral synthesis. In: *IEEE Trans. on CAD* 8 (1989), July, Nr. 6, S. 661–679

**Puschner und Koza 1995** PUSCHNER, P. ; KOZA, C.: Computing Maximum Task Execution Times With Linear Programming Techniques / Technische Universität Wien, Inst. für Technische Informatik. Mai 1995. – Forschungsbericht

**Teich 1997** TEICH, Jürgen: *Digitale Hardware/Software-Systeme: Synthese und Optimierung.* Springer, Berlin, 1997. – Springer-Lehrbuch. – ISBN 3-540-62433-3

**Tseng und Siewiorek 1986** TSENG, C. ; SIEWIOREK, D.P.: Automated synthesis of datapaths in digital systems. In: *IEEE Trans. on CAD* 5 (1986), July, Nr. 3, S. 274–277

**Vivancos u. a. 2001** VIVANCOS, E. ; HEALY, C. ; MUELLER, F. ; WHALLEY, D.: Parametric Timing Analysis. In: *Proceedings of the ACM SIGPLAN Workshop on Language, Compilers, and Tools for Embedded Systems*, URL `http://www.cs.fsu.edu/ whalley/realtimepubs.html`, June 2001

# Anhang A

# Definitionen und Algorithmen

## A.1 Notation von Sequenzgraphen

In der CLINT++ Klassenbeschreibung innerhalb der Wissensbasis werden Sequenzgraphen durch die nachstehende Grammatik in EBNF (*Extended Backus-Naur Form*) beschrieben:

```
cdfg_def := "cdfg" "(" cdfg ")".

cdfg :=  { cdfg_dependency_description |
           cdfg_loop_description       |
           cdfg_branch_description }.

cdfg_dependency_description :=
      "(" cdfg_adjacency_list
      { cdfg_adjacency_list } ")".

cdfg_loop_description :=
      "LOOP" "(" cdfg_upper_loop_bound ")" "(" cdfg ")".

cdfg_branch_description :=
      "BRANCH" "(" cdfg_branch_condition ")" "(" cdfg ")"
             { "(" cdfg_branch_condition ")" "(" cdfg ")" }.

cdfg_adjacency_list :=
      ( cdfg_identifer ":" cdfg_identifier_list |
        cdfg_identifier ) ";".

cdfg_identifier_list := cdfg_identifier { "," cdfg_identifier }.
```

```
cdfg_identifier := ident.

cdfg_upper_loop_bound := integer.

cdfg_branch_condition := integer.
```

Die einzelnen Nichtterminale haben folgende Semantik:

`cdfg_def` Eingangspunkt der CDFG-Syntax für die Einbindung in CLINT++.

`cdfg` steht für einen vollständigen Sequenzgraph.

`cdfg_dependency_description` Beschreibung der Datenabhängigkeiten durch mehrere Adjazenzlisten.

`cdfg_loop_description` Deklaration und Beschreibung eines Schleifenknotens. Der Bezeichner identifiziert den Schleifenknoten in den Adjazenzlisten.

`cdfg_branch_description` Deklaration und Beschreibung eines Verzweigungsknotens. Der Bezeichner identifiziert den Verzweigungsknoten in den Adjazenzlisten.

`cdfg_adjacency_list` Beschreibung von Datenabhängigkeiten. Der erste Bezeichner benennt einen Knoten. In der folgenden Liste werden alle Knoten die von ihm abhängig sind aufgeführt.

`cdfg_identifier_list` Liste der abhängigen Knoten.

`cdfg_identifier` Bezeichner für einen Knoten im Sequenzgraph. Die verwendeten Bezeichner müssen als Parts in der Klassenbeschreibung vorkommen oder in einer LOOP oder BRANCH Beschreibung deklariert worden sein.

`upper_loop_bound` Obere Schranke für die Anzahl von Schleifendurchläufen.

`cdfg_branch_condition` Verzweigungsausdruck. Momentan nicht verwendet. Es wird ein Zahlenwert gelesen der als Verzweigungswahrscheinlichkeit für den folgenden Zweig interpertiert werden kann.

## A.2 Konfigurationsbewertung

Der folgende Algorithmus beschreibt die Berechnung der Konfigurationsbewertung aus den Bewertungen der einzelnen Eigenschaften. Dazu iteriert **fitness** über alle definierten Eigenschaften und ruft für jede die Bewertungsfunktion auf. Je nach Wert des Eingangsparameters werden die verschiedenen Bewertungsfunktionen verwendet.

**Algorithmus 2** Funktion `fitness` zur Bewertung von Konfigurationen.

```
1: fitness(eval_func_type)
2: b_0 := 0
3: ATTR := get_properties(){Eigenschaftsattribute bestimmen}
4: for all attr ∈ ATTR do
5:    attr->evaluate()
6:    req_attr := get_requirement(attr->name()){Anforderung bestimmen}
7:    if eval_func_type = ivl_match then
8:       b := interval_match(req_attr, attr)
9:    else if eval_func_type = ivl_overlap then
10:      b := interval_overlap(req_attr, attr)
11:   else if eval_func_type = ivl_distance then
12:      b := interval_distance(req_attr, attr)
13:   end if
14:   k := get_weight(attr->name());
15:   b_0 := b_0 b^k
16: end for
17: return b_0
```

## A.3 Bewertungsfunktionen

Die Eingangsparameter der Bewertungsfunktionen sind zum Einen das Anforderungsattribut *reqAttr* und das Eigenschaftsattribut *attr*. Die Subscripte *min* und *max* bezeichnen jeweils die untere und die obere Intervallgrenze der Attribute.

**Algorithmus 3** Bewertungsfunktion `interval_match`.

1: `interval_match`($reqAttr$,$attr$)
2: **if** $attr_{min} \geq reqAttr_{min} \wedge attr_{max} \leq reqAttr_{max}$ **then**
3: **return** 1
4: **else**
5: **return** 0
6: **end if**

**Algorithmus 4** Bewertungsfunktion `interval_overlap`.

1: `interval_overlap`($reqAttr$,$attr$)
2: **if** $attr_{min} = attr_{max}$ **then**
3: **if** $attr_{min} \geq reqAttr_{min} \wedge attr_{min} \leq reqAttr_{max}$ **then**
4: $b$:=1
5: **else**
6: $b$:=0
7: **end if**
8: **else**
9: $overlap$:=$\min(reqAttr_{max}, attr_{max}) - \max(reqAttr_{min}, attr_{min})$
10: **if** $overlap \geq 0$ **then**
11: $b$:=$\frac{overlap}{attr_{max} - attr_{min}}$
12: **else**
13: $b$:=0
14: **end if**
15: **end if**
16: **return** $b$

**Algorithmus 5** Bewertungsfunktion `interval_distance`.

1: `interval_distance`($reqAttr$,$attr$)
2: $overlap$:=$\min(reqAttr_{max}, attr_{max}) - \max(reqAttr_{min}, attr_{min})$
3: **if** $overlap \geq 0$ **then**
4: $b$:=1
5: **else**
6: $b$:=$\frac{attr_{max} - attr_{min}}{attr_{max} - attr_{min} + \max(reqAttr_{min} - attr_{max}, attr_{min} - reqAttr_{max})}$
7: **end if**
8: **return** $b_0$

## A.4 Ermittlung der WCET aus der Wissensbasis

Die WCET-Analyse besteht darin, die in der Wissensbasis gespeicherten Informationen über die WCET von Basisfunktionen auf bestimmten Ressourcen auszuwerten. Der Algorithmus 6 benutzt die entsprechenden Attribute (s. Tabelle 3.2) der Basisfunktionen. Der Eingangsparameter func muss eine Basisfunktion sein. Ist func an eine Ressource gebunden, so sucht der Algorithmus nach einem Attribut, dessen Name sich aus "WCET_" und dem Namen der Ressource zusammensetzt (Zeilen 3-6). Ist func nicht gebunden, so ermittelt der Algorithmus die maximal mögliche WCET unter allen Bindungsmöglichkeiten (Zeilen 8-13).

**Algorithmus 6** Ermittlung der WCET aus der Wissensbasis.

```
 1: wcet WCET_Analyse(func)
 2: wcet = 0
 3: if func ist an eine Ressource gebunden then
 4:    res := Bindung(func)
 5:    attr := func.getAttr("WCET_" + res.name)
 6:    wcet := attr_max
 7: else
 8:    for all attr ∈ func.attrs do
 9:       if attr.name beginnt mit "WCET" then
10:          wcet = max(wcet, attr_max)
11:       end if
12:    end for
13: end if
14: return wcet
```

## A.5 Attributmethoden

**Algorithmus 7** Attributmethode "WCET".

1: success `eval_wcet`(comp,*attr*)
2: top := Topklasse_Funktionsschicht()
3: resetVisitedCDFG(top)
4: $min$ := SCHEDULE(top,true)
5: $max$ := SCHEDULE(top,false)
6: **if** $[min, max] \subset [attr_{min}, attr_{max}]$ **then**
7: $attr := [min, max]$
8: **end if**
9: **return** true

**Algorithmus 8** Attributmethode "COST".

```
 1: success eval_cost(comp,attr)
 2: attr := [0, 0, 0]
 3: atop := Topklassse_Architekturschicht()
 4: if Existiert(atop) then
 5:    a := func.get_attr(attr.name + impl.name)
 6:    attr_min := attr_min + a_min
 7:    attr_val := attr_val + a_val
 8:    attr_max := attr_max + a_max
 9:    for all comp ∈ Parts von atop do
10:       impl := Blattklasse unter comp
11:       for all func ∈ an comp gebundene Funktionen do
12:          a := func.get_attr(attr.name + impl.name)
13:          attr_min := attr_min + a_min
14:          attr_val := attr_val + a_val
15:          attr_max := attr_max + a_max
16:       end for
17:       for all func ∈ ungebundene Funktionen do
18:          for all a ∈ func.attrs do
19:             if a.name = attr.name + impl.name then
20:                attr_min := attr_min + a_min
21:                attr_val := attr_val + a_val
22:                attr_max := attr_max + a_max
23:             end if
24:          end for
25:       end for
26:    end for
27:    return true
28: else
29:    return false
30: end if
```

# Anhang B

# Wissensbasis des Fallbeispiels

Für das Fallbeispiel wurde die Wissensbasis videocodec.clint. entwickelt. Die beschreibt mit Hilfe der Sprache CLINT++ (*CLass and Instance NoTation*) einen Kodierer/Dekodierer für Bildsequenzen nach der Norm H.261, ein System aus der Domäne der digitalen Signalverarbeitung. Im Folgenden wird der Inhalt der Wissensbasis kurz erläutert.

## B.1 Spezifikationsschicht

Die Spezifikationsschicht ermöglicht die Spezifikation von drei verschiedenen Systemen. Diese unterscheiden sich in ihrer Funktionalität. Es sind ein Decoder, ein Encoder oder ein Codec, welcher Kodierung und Dekodierung beherrscht, auswählbar. Als nichtfunktionale Anforderungen wurden die Ausführungszeit "WCET", die Kosten "COST" und der Energieverbrauch "PWR" einbezogen. Der Spezifikationseditor erlaubt die Spezifikation dieser Anforderungen. Um die entsprechenden Schlüsselfunktionen[1] der Funktionsschicht zu instanzieren, beinhaltet die Spezifikationsschicht drei Relationen vom Typ "multirel".

[1] Keyfunctions

## B.2 Funktionsschicht

Die Funktionalität des Videocodec ist in 15 Basisfunktionen unterteilt. Eine Übersicht über die Basisfunktionen liefert Tabelle B.1. Eine grafische Darstellung der Funktionsschicht ist in Abbildung B.1 zu sehen. Jede Basisfunktion kann auf verschiedene Ressourcen gebunden werden. Alle Bindungsmöglichkeiten sind in Tabelle B.2 zusammengefasst.

## B.3 Architekturschicht

In der Architekturschicht sind drei Architekturen vorhanden. Alle Ressourcen sind über einen gemeinsamen Bus verbunden. Die Ressourcen "A_IN", "A_OUT" sowie der Bildspeicher "A_FM" sind in jeder Architektur vorhanden, da sie für einige Basisfunktionen essentiell sind. Die Architektur "ARCH_TOP_0" besitzt darüber hinaus nur eine Recheneinheit in Form eines Prozessors. In Architektur "ARCH_TOP_1" existieren für jede Basisfunktion spezielle Hardwaremodule, welche die Funktionalität implementieren. Die Architektur "ARCH_TOP_2" stellt einen Kompromiss zwischen spezieller Hardware und allgemeinen Recheneinheiten dar. Sie verfügt über einen Prozessor und einen Signalprozessor. Abbildungen B.2 - B.4 zeigen Blockschaltbilder der Architekturen.

## B.4 Implementationsschicht

In der Implementationsschicht befinden sich alle Ressourcen, auf die die Basisfunktionen gebunden werden können. Dazu gehören spezielle Module, welche nur bestimmte Funktionen ausführen können, und allgemeinere Recheneinheiten, die beliebige Funktionalität implementieren können. Eine Übersicht über die in der Wissensbasis vorhandenen Ressourcen befindet sich in Tabelle B.3

| Basisfkt. | Bemerkung |
|---|---|
| IN | Einlesen eines Datenpaketes |
| OUT | Ausgabe eines Datenpaketes |
| BM | Bewegungsschätzung (Block Matching) |
| RF | Lesen eines Bildes aus dem Bildspeicher (Read Frame) |
| SF | Speichern eines Bildes in den Bildspeicher (Store Frame) |
| LF | Filteroperation |
| DIFF | Subtraktion zweier Bildblöcke |
| DCT | Diskrete Cosinustransformation eines Bildblockes |
| IDCT | Inverse diskrete Cosinustransformation eines Bildblockes |
| TH | Schwellwert (Threshold) |
| Q | Quantisierung |
| IQ | Inverse Quantisierung |
| SUM | Addition zweier Bildblöcke |
| RLC | Lauflängenkodierung (Run Length Coding) |
| RLF | Lauflängendekodierung (Run Length Decoding) |

Tabelle B.1: Übersicht der Basisfunktionen.

| Basisfkt. | Ressource/WCET |
|---|---|
| IN | H_IN/0 |
| OUT | H_OUT/0 |
| BM | H_BM_FPGA/34, H_BM_GA/22, H_TMS320/60, H_ARM/102, H_SPARC/85, H_PII300/153, H_PII600/98 |
| RF | H_MEMORY/0, H_DPMEMORY/0 |
| SF | H_MEMORY/0, H_DPMEMORY/0 |
| LF | H_HC/2, H_TMS320/3, H_ARM/11, H_SPARC/9, H_PII300/20, H_PII600/10 |
| DIFF | H_SAM/1, H_TMS320/1, H_ARM/2, H_SPARC/2, H_PII300/4, H_PII600/2 |
| DCT | H_DCT/2, H_TMS320/4, H_ARM/10, H_SPARC/8, H_PII300/12, H_PII600/6 |
| IDCT | H_DCT/2, H_TMS320/4, H_ARM/10, H_SPARC/8, H_PII300/12, H_PII600/6 |
| TH | H_HC/2, H_TMS320/8, H_ARM/10, H_SPARC/8, H_PII300/16, H_PII600/8 |
| Q | H_HC/1, H_TMS320/2, H_ARM/2, H_SPARC/2, H_PII300/4, H_PII600/2 |
| IQ | H_HC/1, H_TMS320/2, H_ARM/2, H_SPARC/2, H_PII300/4, H_PII600/2 |
| SUM | H_SAM/1, H_TMS320/1, H_ARM/2, H_SPARC/2, H_PII300/4, H_PII600/2 |
| RLC | H_HC/2, H_TMS320/8, H_ARM/10, H_SPARC/8, H_PII300/16, H_PII600/8 |
| RLD | H_HC/2, H_TMS320/8, H_ARM/10, H_SPARC/8, H_PII300/16, H_PII600/8 |

Tabelle B.2: Bindungsmöglichkeiten der Basisfunktionen.

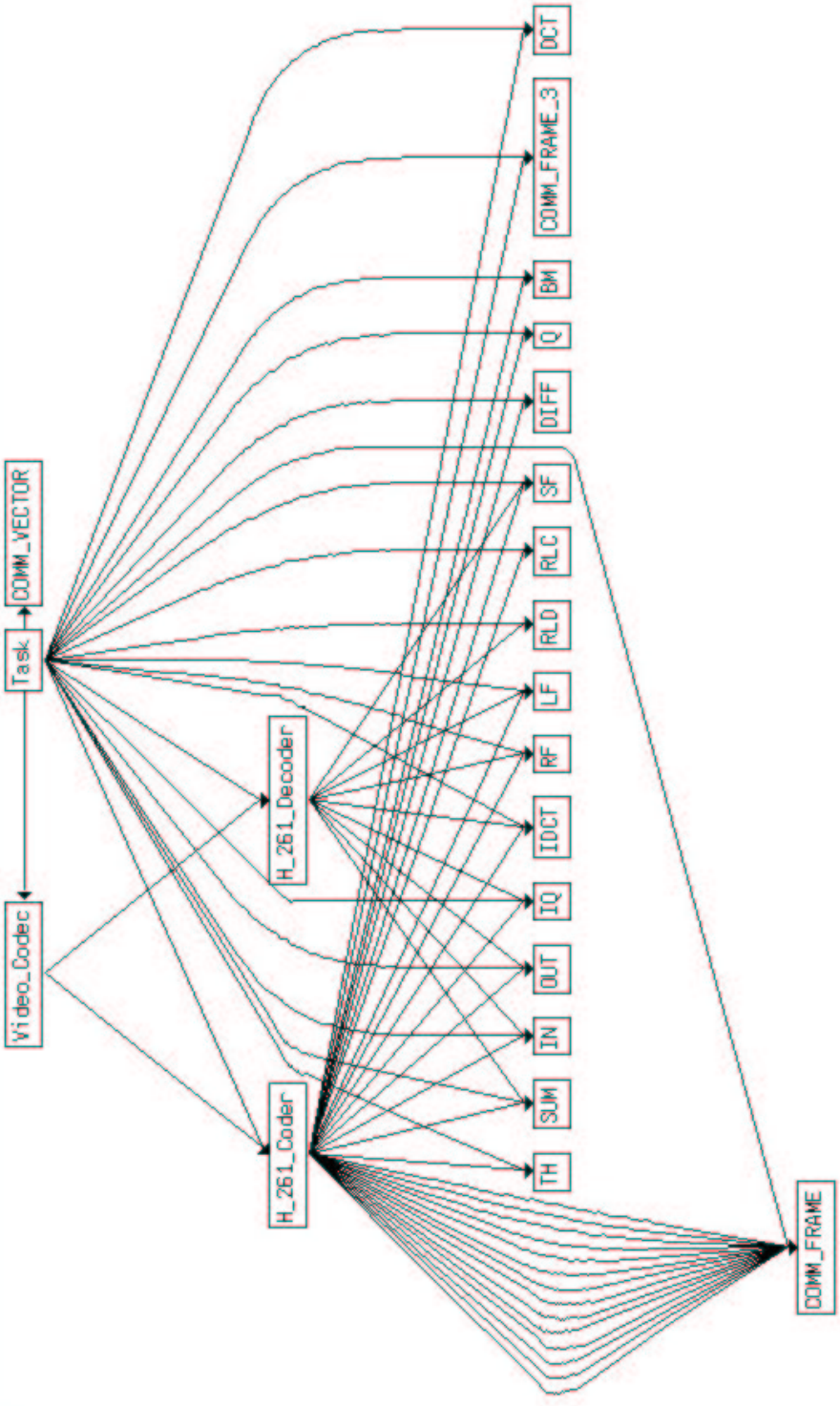


Abbildung B.1: Funktionsschicht der Wissensbasis videocodec.clint.

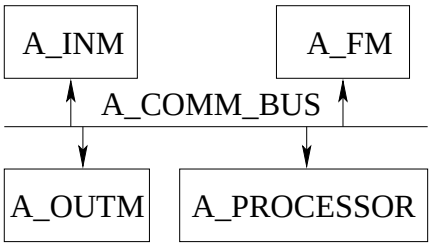


Abbildung B.2: Architektur “ARCH_TOP_0”.

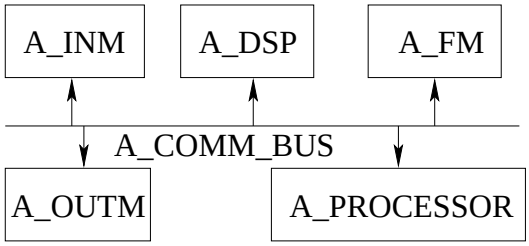


Abbildung B.3: Architektur “ARCH_TOP_1”.

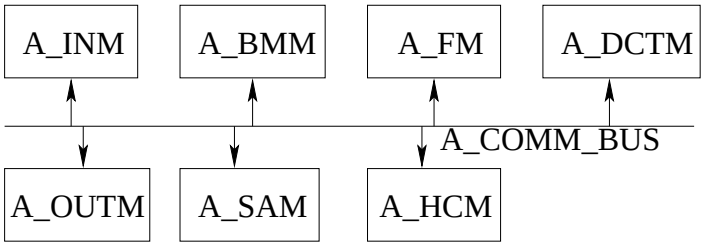


Abbildung B.4: Architektur “ARCH_TOP_2”.

| Ressource | COST | PWR |
|---|---|---|
| H_ARM | 125 | 10 |
| H_SPARC | 152 | 13 |
| H_PII300 | 100 | 18 |
| H_PII600 | 150 | 25 |
| H_8051 | 18 | 2 |
| H_68HC11 | 10 | 1 |
| H_TMS320 | 200 | 5 |
| H_BM_FPGA | 120 | 2 |
| H_BM_GA | 90 | 1.5 |
| H_IN | 3 | 0.2 |
| H_OUT | 3 | 0.2 |
| H_BUS1 | 2-4 | 0.1-0.14 |
| H_BUS2 | 3-6 | 0.15-0.18 |
| H_BUS3 | 5-7 | 0.2-0.25 |
| H_MEMORY | 20 | 0.2 |
| H_DPMEMORY | 40 | 0.3 |
| H_HC | 50 | 0.6 |
| H_DCT | 100 | 1.5 |
| H_SAM | 50 | 0.8 |

Tabelle B.3: Ressourcen der Implementationsschicht.

# Glossar

**Architekturschicht** Die Architekturschicht dient dem Management geteilter Ressourcen. Sie beschreibt, welche Klassen der Funktionsschicht eine gemeinsame Ressource nutzen sollen (z.B. Abarbeitung mehrerer Verarbeitungsfunktionen auf dem gleichen Prozessor, Abbildung mehrerer Kommunikationsfunktionen auf dem gleichen Bus).

**Aussage** Eine Aussageform wird zur Aussage wenn den in ihr vorkommenden freien Variablen Werte zugeordnet werden. Bsp: 7 teilt 35, $12 < 10$. Aussagen unterscheiden sich von Aussageformen dadurch, dass ihnen ein Wahrheitswert zugrordnet werden kann.

**Aussageform** Eine Aussageform ist ein sprachlicher oder symbolischer Ausdruck, in dem mindestens eine Leerstelle (freie Variable) vorkommt. Bsp: $x$ teilt 35, $a < 10$.

**Eingebettete Systeme** Eingebettete Systeme bilden jene echte Teilmenge aller Systeme, deren Elemente informationserfassende, -übertragende, -verarbeitende, -speichernde oder steuernde Einheiten sind. Eingebettete Systeme in ihrer Gesamtheit sind Elemente eines umfassenderen Systems. Sie haben meist reaktiven Charakter. Der Systemrand des eingebetteten Systems bildet die Schnittstelle zum Wirtssystem, an der der bidirektionale Informationsaustausch zwischen Elementen des Wirtssystems und denen des eingebetteten Systems stattfindet. Als Elemente eingebetteter Systeme dürfen eingebettete Systeme auftreten (hierarchische Struktur).

**Funktionsschicht** Die Funktionsschicht beschreibt die zur Realisierung der

Use Cases (definiert in der Spezifikationsschicht erforderlichen (Teil-) Funktionen auf implementierungsneutralem Niveau (d.h. unabhängig von einer Entscheidung über die Implementierung in Hard- oder Software). Den Klassen der Funktionsschicht werden ausführbare Verhaltensbeschreibungen (z.B. in Form von C-Quelltexten) zugeordnet, welche die Generierung eines Prototyps des Zielsystems ermöglichen.

**Geteilte Ressource** Eine Ressource bezeichnet man als Geteilte Ressource, wenn sie von mehreren Prozessen zur gleichen Zeit benötigt wird. Die Zuteilung der Ressourcen erfolgt dezental (durch die Prozesse selbst organisiert) oder zentral (in einer Übergeordneten Instanz, Scheduling). Die Zuteilung muß sicherstellen, dass keine Verklemmungen (Deadlocks) entstehen.

**Implementationsschicht** Die Implementationsschicht verwaltet die physischen Ressourcen, die am Ziel des Entwurfsprozesses stehen. Im Falle von Hardware sind dies simulier- bzw. synthetisierbare Beschreibungen von Komponenten (Mikroprozessoren, Mikrocontroller, ASIC, ASIP, FPGA, PLD usw. in einer Hardware-Beschreibungssprache (z.B. VHDL). Im Falle von Software sind dies Programme im Quelltext-Format einer imperativen Programmiersprache (z.B. C) oder im Maschinencode eines Spezialprozessors.

**Instanz** Instanzen sind (Software-) Objekte, die während der Konfiguration eines Systems aus Klassen abgeleitet und mit Werten angereichert werden, um reale Komponenten mit hinreichender Genauigkeit beschreiben zu können.

**Konfiguration** Eine Konfiguration ist der im Ergebnis manueller oder automatischer Entwurfsschritte ausgewählte Weg durch die Hierarchiegraphen der Wissensbasis. Auf Grund der Relationen bilden die Klassen einen Instanzenbaum. Die Beschreibung einer Konfiguration erfolgt ebenfalls mit den Mitteln der Beschreibungssprache CLINT++ durch Angabe des Namens der Instanz und des Namens der Klasse, aus der diese Instanz gebildet wurde. Damit ist der Zugriff auf alle übrigen,

während des Entwurfsprozesses unverändert gebliebenen Bestandteile der Klasse möglich (insbesondere die Verhaltensbeschreibung der Klasse). Ferner enthält die Konfiguration Werte, die den Attributen zugewiesen wurden. Wesentlich für die Generierung des ausführbaren Modells sind die Instanzen der Implementierungsschicht. Jede dieser Instanzen beschreibt genau eine Komponente.

**Ressource** Der Begriff der Ressource wird innerhalb von RODOS synonym für ein Betriebsmittel verwendet. In der Betriebssystem-Theorie für konventionelle Rechnersysteme gilt eine Ressource als geteilt, wenn sie von mehr als einem ProzeSS nutzbar ist. Im Bereich der eingebetteten Systeme gilt darüber hinaus eine Ressource als geteilt, wenn mehr als eine Klasse der Funktionsschicht auf eine Komponente der Implementierungsschicht abgebildet wird. Die Art der Abbildung wird in der Architekturschicht beschrieben. Dabei ist es unerheblich, ob z.B. zwei Programme sequentiell auf einem Prozessor abgearbeitet werden oder ob dies im Time-Sharing-Betrieb (unter Steuerung eines RTOS) erfolgt.

**Spezifikationsschicht** Die Spezifikationsschicht nimmt alle funktionalen und nicht-funktionalen Anforderungen an das zu entwerfende System auf. Eine Möglichkeit zu Spezifikation funktionaler Anforderung besteht in der Beschreibung von Use Cases durch UML und deren anschlieSSender Konvertierung in Klassen der Spezifikationsschicht. Nicht-funktionale Anforderungen sind z.B. Kosten, Reaktionszeit, Energiebedarf, Zuverlässigkeit usw. Sie werden durch Intervalle von Attributwerten beschrieben, welche die Anforderungen des Nutzers an das System darstellen.

**Wissensbasis** Eine Wissensbasis ist die Gesamtheit aller über den Diskursbereich verfügbaren Informationen. Sie stellt Container zur Aufnahme von Faktenwissen bereit und verwaltet Methoden zur Bearbeitung dieses Wissens. Sie ist ein informationstechnisches Modell, dessen Elemente nicht nur die Elemente des realen Systems sondern auch die zu seiner

Spezifikation und Implementierung notwendigen Informationen abbilden. Eine Wissensbasis, die den Entwurfsprozess eingebetteter Systeme unterstützt, soll ein kongruentes und konsistentes Abbild dieser Domäne sein. Sie besteht aus der Datenbasis und einer Menge von Prüf- und SchluSSregeln über den Wissenseinheiten. Die Wissensbasis von RODOS enthält mehrere logische Schichten.

**Wissenseinheit** Wissenseinheiten sind (in Analogie zur kybernetischen Systemtheorie) Beschreibungen von Systemelementen durch Klassen sowie Beschreibungen von Beziehungen zwischen diesen durch Relationen). Jede Klasse der Wissensbasis gehört genau einer logischen Schicht an. Die Wissenseinheit ist die kleinste adressierbare Aggregation von Fakten und/oder Beziehungen aus der betrachteten Domäne. Die Zerlegung des gesamten verfügbaren Wissens in Wissenseinheiten ist unabhängig von technischen Systemen für deren Speicherung oder Verarbeitung. Wissenseinheiten im o.g. Sinne vertreten nicht nur die Komponenten des zu entwerfenden Zielsystems sondern auch alle auf höheren Abstraktionsstufen entwickelten Modelle des Systems (z.B. Spezifikationsmodell, Funktionsmodell, Architekturmodell). Zur textuellen Repräsentation der Wissenseinheiten und ihrer Beziehungen steht die deskriptive Sprache CLINT++ zur Verfügung.

# Index

# Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit allein und unter Verwendung der angegebenen Literatur angefertigt, sowie Zitate kenntlich gemacht zu haben.

Marco Fischer Chemnitz, 27. Februar 2002